UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DEL JUEGO "BUBBLE SHOOTER" MEDIANTE XNA 3.1

Autor. Marcos Carro Arribas

Tutor. Juan Peralta Donate

Febrero, 2011

Agradecimientos

En esta sección me gustaría dar las gracias a todas las personas que me han ayudado desde que empecé la carrera hasta el día de hoy a poder realizar la presentación de este Proyecto Fin de Carrera.

Quiero comenzar dando especialmente las gracias a mis padres, Emilia y Emiliano, por los sacrificios hechos y el apoyo que he recibido para ayudarme siempre con mis estudios. Por haber creído siempre en mí, por preocuparse día a día y respaldarme cuando lo he necesitado. Gracias también a mis hermanos, Eva y Javier, por estar siempre ahí y echarme una mano siempre que en alguna ocasión he necesitado que me echaran una mano.

A mis compañeros de universidad. En especial a Almudena, Javi, Jorge, Miguel y Víctor, por sus apoyos y amistad durante todos estos años y nuestras partiditas entre las clases.

A mis amigos, por soportar todas mis batallitas de informático.

A Juan Peralta, mi tutor, por darme la oportunidad de realizar este precioso proyecto. Por toda la ayuda y tiempo dedicado. Siempre ha estado disponible para resolver dudas y aportar sugerencias y conocimientos. Sin su colaboración, este proyecto nunca lo habría conseguido hacer.

Al tribunal, por aceptar la invitación a la lectura de este Proyecto Fin de Carrera.

Por último y muy especialmente, a Montse. Por ayudarme con las responsabilidades y así haber tenido más tiempo para realizar el Proyecto Fin de Carrera. Por apoyarme en todos los momento de dificultad y conseguir que afronte todo con ilusión. Gracias por estar siempre ahí.

En definitiva, gracias a todos los que me han ayudado a hacer realidad mis sueños.

Índice general

Agradecimientos **3**
Índice general **5**
Índice de figuras **7**
Índice de código **10**
Índice de tablas **11**
Capítulo 1 ..... **13**
Introducción **13**
1.1. Descripción general 14
1.2. Motivación del Proyecto 15
1.3. Objetivos del Proyecto 15
1.4. Contenido de la memoria 18
Capítulo 2 ..... **19**
Estado de la cuestión **19**
2.1. Antecedentes 20
2.2. Historia de los videojuegos 20
2.2.1. Orígenes 21
2.2.2. Evolución de los videojuegos 22
2.2.3. Los videojuegos en la actualidad 60
2.2.4. Futuro de los videojuegos 63
2.3. Ciclo de vida de los videojuegos 67
2.3.1. Concepción de la idea 67
2.3.2. Diseño 70
2.3.3. Planificación 71
2.3.4. Producción 72
2.3.5. Pruebas 73
2.3.6. Mantenimiento 74
2.4. XNA Game Studio 75
2.4.1. Qué es XNA y por qué usarlo 75
2.4.2. Arquitectura de XNA Framework 76
2.4.3. XNA Build 81
2.4.4. XNA Game Studio 83
2.5. Otras arquitecturas 84
Capítulo 3 ..... **89**
Gestión del proyecto **89**
3.1. Fase de Análisis 90
3.1.1. Idea inicial 90
3.1.2. Identificación de requisitos 91
3.1.3. Especificación de casos de uso 97
3.1.4. Diagrama de actividad del sistema 102
3.1.5. Diagramas de secuencia 102
3.2. Fase de Diseño 107
3.2.1. Diagrama de clases 107
3.2.2. Definición de las clases 108
3.3. Fase de Implementación 118
3.3.1. Elementos que intervienen en el juego 119
3.3.2. Algoritmos 130
3.3.3. Sistema de puntuación 137
3.3.4. Niveles 137
Capítulo 4 ..... **139**
Objetivos **139**
4.1. Objetivos y problemas encontrados 140
4.2. Conclusiones 141

Capítulo 5 ..... **143**
Líneas futuras **143**
5.1. Futuros trabajos a realizar 144
Anexo A **145**
Planificación **145**
A1. Introducción 146
A2. Ciclo de vida 146
A3. Planificación del proyecto 148
A4. Presupuesto del proyecto 159
A.4.1. Estimación 159
A.4.2. Presupuesto final 160
A5. Publicación del juego 160
Diccionario de términos y acrónimos **163**
Referencias **169**

Índice de figuras

FIGURA 1: Skill Shooter……..20
FIGURA 2: Lanzamiento de misiles…….21
FIGURA 3: Máquina del juego Pong……...22
FIGURA 4: Space War…….23
FIGURA 5: Nintendo Entertainment System……...24
FIGURA 6: Master System……...25
FIGURA 7: Atari 7800…….25
FIGURA 8: MegaDrive……26
FIGURA 9: Sinclair ZX Spectrum……...26
FIGURA 10: Amstrad CPC 464……...27
FIGURA 11: Pantalla del Donkey Kong……..27
FIGURA 12: Pantalla del Haunted House……28
FIGURA 13: Pantalla inicial del Pacman…….28
FIGURA 14: Fase del juego Dragon's Lair……..29
FIGURA 15: Pantalla del juego Mario Bros…….29
FIGURA 16: Carátula del juego Final Fantasy…….30
FIGURA 17: Portada de la revista MicroHobby……..31
FIGURA 18: Fase del castillo de King's Quest I…….31
FIGURA 19: Super Mario Bros……32
FIGURA 20: Pantalla de Tetris…….32
FIGURA 21: Pantalla inicial del Out Run……33
FIGURA 22: Pantalla del juego Arkanoid……33
FIGURA 23: Fase del juego Metal Gear……..34
FIGURA 24: La abadía del crimen……...35
FIGURA 25: Juego Sim City...…….35
FIGURA 26: Pang…….36
FIGURA 27: SNES……...36
FIGURA 28: Game Gear y Game Boy…….37
FIGURA 29: Neo Geo……..37
FIGURA 30: Saturn……..38
FIGURA 31: PlayStation……..39
FIGURA 32: Nintendo 64…….40
FIGURA 33: DreamCast……...40
FIGURA 34: Escena de Sonic……..41
FIGURA 35: Escena de Street Fighter II……..42

FIGURA 36: Acción Fatality de Mortal Kombat…….42
FIGURA 37: Escena 3D de Alone in the Dark…….43
FIGURA 38: Imagen del FIFA 94……44
FIGURA 39: Escena de Resident Evil……..45
FIGURA 40: Escena de Age of Empires……..46
FIGURA 41: Escena de Grand Theft Auto……...46
FIGURA 42: Imagen de Gran Turismo……47
FIGURA 43: Escena de tiroteo en Half Life……48
FIGURA 44: Escena de StarCraft…….49
FIGURA 45: Portada de Commandos……..49
FIGURA 46: PlayStation 2 y mando……50
FIGURA 47: Xbox de Microsoft……..51
FIGURA 48: Nintendo DS……52
FIGURA 49: PlayStation Portable……52
FIGURA 50: Xbox 360…….53
FIGURA 51: Wii de Nintendo……..54
FIGURA 52: PlayStation 3 de Sony…….54
FIGURA 53: Escena de batalla en Halo……...55
FIGURA 54: Violencia en GTA San Andreas……..56
FIGURA 55: Carátula de Metal Gear Solid 3……...57
FIGURA 56: Escena de God of War……58
FIGURA 57: Accesorios de Guitar Hero……..58
FIGURA 58: Secuencia de Super Mario Galaxy……..59
FIGURA 59: Componentes de Kinect……..61
FIGURA 60: VirtualBoy de Nintendo……..64
FIGURA 61: Imagen de Second Life……...66
FIGURA 62: Guión gráfico del anuncio "*La fábrica de la felicidad*" de Coca-Cola……...70
FIGURA 63: Diagrama de GANTT…….72
FIGURA 64: Diagrama de arquitectura del Framework de XNA.....77
FIGURA 65: Capas del Framework de XNA……...78
FIGURA 66: Content Pipeline en tiempo de compilación……...82
FIGURA 67: Content Pipeline en tiempo de ejecución……82
FIGURA 68: Diagrama de casos de uso……...98
FIGURA 69: Diagrama de actividad del sistema……...102
FIGURA 70: Diagrama de secuencia de Jugar……...104
FIGURA 71: Diagrama de secuencia de Dibujar……...105
FIGURA 72: Diagrama de secuencia del disparo de una bola……...105

FIGURA 73: Diagrama de secuencia del movimiento de una bola disparada……...106
FIGURA 74: Diagrama de secuencia del proceso de eliminación de bolas……...106
FIGURA 75: Diagrama de clases……...108
FIGURA 76: Detalle de las clases del paquete ScreenManager…….110
FIGURA 77: Definición de la clase MenuScreen……...112
FIGURA 78: Definición de la clase PlayScreen…….114
FIGURA 79: Definición de la clase ObjetosJuego…….117
FIGURA 80: Texturas del lanzador……120
FIGURA 81: Texturas de las bolas…….120
FIGURA 82: Rectángulo de colisiones……...124
FIGURA 83: Colisión de dos bolas……126
FIGURA 84: División del centro de masas de una bola…… 126
FIGURA 85: Comparación de bolas adyacentes……133
FIGURA 86: Supuesto de una colisión……...133
FIGURA 87: Tareas del diagrama de GANTT inicial ……...152
FIGURA 88: Tareas del diagrama de GANTT real…..……...153
FIGURA 89: Diagrama de GANTT real de fase formación previa……155
FIGURA 90: Diagrama de GANTT inicial de fase formación previa……155
FIGURA 91: Diagrama de GANTT real de fase estructural del proyecto…….156
FIGURA 92: Diagrama de GANTT inicial de fase estructural del proyecto…….156
FIGURA 93: Diagrama de GANTT real de fase de creación del juego I...……156
FIGURA 94: Diagrama de GANTT inicial de fase de creación del juego I...……156
FIGURA 95: Diagrama de GANTT real de fase de creación del juego II..……...157
FIGURA 96: Diagrama de GANTT inicial de fase de creación del juego II..……...157
FIGURA 97: Diagrama de GANTT real de fase de gestión de pantallas……...157
FIGURA 98: Diagrama de GANTT inicial de fase de gestión de pantallas……...157
FIGURA 99: Diagrama de GANTT real de fase de elementos extra…..……...158
FIGURA 100: Diagrama de GANTT inicial de fase de elementos extra…..…….158

Índice de código

CÓDIGO 1: Texturas de los elementos del juego……120
CÓDIGO 2: Centro de rotación del lanzador……121
CÓDIGO 3: Rotación del lanzador……...121
CÓDIGO 4: Función del momento de disparo de un cohete……122
CÓDIGO 5: Movimiento de un cohete……123
CÓDIGO 6: Dibujo del movimiento de un sprite…….....124
CÓDIGO 7: Detección de colisiones……125
CÓDIGO 8: Intercepción de objetos BoundingBox……126
CÓDIGO 9: Inicialización de la matriz de objetos……...128
CÓDIGO 10: Movimiento del lanzador a través del ratón……...129
CÓDIGO 11: Definición de objetos de efectos de sonido……130
CÓDIGO 12: Algoritmo de búsqueda de nodos……...132
CÓDIGO 13: Destrucción de nodos……135
CÓDIGO 14: Actualización del juego ……136

Índice de tablas

TABLA 1: Generaciones de consolas……..55
TABLA 2: Lista ALGeneral……..134
TABLA 3: Tareas del proyecto inicial…..……..150
TABLA 4: Tareas del proyecto real…..……..151
TABLA 5: Recursos y horas asociados al proyecto……..159
TABLA 6: Costes de los recursos asociados al proyecto……..160
TABLA 7: Presupuesto final del proyecto…….160

Capítulo 1

Introducción

Esta sección representa un enfoque general de los propósitos de este proyecto. Los puntos a destacar son la motivación del proyecto, los objetivos y los contenidos de la memoria.

El segundo apartado de este capítulo, Motivación del proyecto, representa los motivos por los que se ha hecho este proyecto.

El tercer apartado, Objetivos, expone los objetivos que vamos a conseguir con la realización del proyecto.

Por último en el cuarto, Contenido de la memoria, se introducen los contenidos principales del proyecto aportando una visión general de este.

1.1. Descripción general

Los *videojuegos*[1] constituyen una de las modalidades encuadradas en el ámbito de las Tecnologías de la Informática y las Comunicaciones (*TIC*). En los últimos 30 años, esta modalidad informática ha sufrido un aumento tanto del interés como en la demanda por parte, sobre todo de los adolescentes, aunque cada vez más son los adultos los que compran videojuegos.

Desde que existe la informática, son muchos los que han intentado usarla como una distracción, y que mejor manera que creando juegos. Hace 3 décadas se conseguía hacer juegos con una calidad aceptable programando en Basic. El éxito fue evidente, y con lenguajes de programación más sofisticados como C y C++ se consiguieron juegos con muy buenas calidades. Tal ha sido el éxito que durante los últimos años han nacido empresas dedicadas exclusivamente a hacer videojuegos, con un resultado sobresaliente. Este mercado ha crecido de una manera extraordinaria, y cada año salen al mercado títulos a los que se les ha dedicado tanto o más tiempo que a una superproducción de Hollywood.

En este proyecto, se ha pretendido diseñar y desarrollar un videojuego mediante una de las tecnologías más punteras en el mercado de los videojuegos como base, el *API*[2] de *Microsoft XNA*[3], y la consola *XBOX 360*[4] y el PC como plataformas finales. El juego elegido es el "*Bubble shooter*"[5], que tanta expectación y fieles captó durante mucho tiempo, y que hoy en día es uno de los juegos más buscados en los portales dedicados a juegos en línea. Esta API de Microsoft, representa la base del software para implementar juegos para Xbox. Más adelante se explicará qué es, cuáles son sus cualidades, puntos fuertes, etc., para entender más y mejor el mundo de los videojuegos.

Bubble shooter [1] es uno de los juegos más adictivos de todos los tiempos. Aunque existen muchas variaciones, todos ellos se basan en el juego original de *Puzzle Bubble*. *Puzzle Bubble*, fue desarrollado originalmente por *Taito* y publicado por *Sega* en 1994. El juego combina elementos de acción, puzle y estrategia.

Como novedad en este proyecto, se ha elaborado un juego con reglas ligeramente distintas a las del juego original, pero sin perder la esencia del mismo. El sistema de puntuación, la opción de jugar con 3 niveles de dificultad, hacer que el juego se complique a medida que se suman puntos, la elección de desarrollar un algoritmo propio para la búsqueda y eliminación de objetos iguales cuando colisionan, son ejemplos de las novedades que el juego ofrece.

1.2. Motivación del Proyecto

Lo que empezó siendo un pasatiempo para unos pocos hace 50 años, hoy en día se ha convertido en uno de los negocios más rentables de la era contemporánea, y que no solo no padece la crisis actual, sino que cada año las empresas dedicadas a este sector incrementan sus beneficios; este fructuoso negocio es el de los videojuegos. Tanto es así, que hace unos años se hacían videojuegos basados en películas, pero el auge del mercado ha hecho girar la flecha al sentido contrario, y son muchos los directores de Hollywood quienes dedican grandes cantidades de dinero a crear películas basadas en títulos de videojuegos con tanto éxito como *Tomb Raider*, *Resident Evil*, y el más actual, *Prince of Persia*.

Está demostrado que los videojuegos influyen en las destrezas cognitivas de los usuarios, en la coordinación viso-motriz y en la velocidad de reacción ante un estímulo determinado [2]. Jugar con videojuegos implica concentración, atención, control y sobre todo, emoción. Cada día se recomienda a los padres que sus hijos consuman videojuegos, siempre dentro de unos límites sanos.

El mundo de los videojuegos ha cambiado mucho desde su origen a la actualidad, y se ha expandido para evitar los descensos de ventas y mantener el interés del público, que ya no se reduce a niños y adolescentes, sino que cada vez trata de abarcar a todas las edades. En los comienzos, el modelo de ventas se basaba en la salida de las nuevas *consolas*[6] y caía demasiado rápido hasta la salida de una nueva generación. Ahora los nuevos dispositivos ponen al servicio del usuario juegos de muchos tipos, abriendo el mercado a usuarios que nunca habían jugado a los videojuegos, este es el caso de personas mayores que con juegos como el *Brain training* de *Nintendo* han entrado de manera muy fuerte a adquirir este tipo de producto.

1.3. Objetivos del Proyecto

Los objetivos principales planteados en este proyecto fin de carrera son los siguientes:

- Desarrollar sin tener conocimientos previos de la tecnología aplicada, un videojuego utilizando la herramienta *Microsoft Visual Studio .NET* y el API para desarrollo de videojuegos de *Microsoft XNA 3.1*.
- Conocer y aplicar al juego la mayor cantidad de las posibilidades que ofrece el diseño de videojuego; empezando por algo tan básico como crear un *sprite*[7], pasando por mover los objetos por pantalla, y posibilidades más avanzadas como la detección de colisiones, y realizar acciones cuando esto ocurre.

- Añadir funcionalidades al juego original, tales como la elección del nivel de juego, incrementar el nivel de dificultad según se van consiguiendo puntos y crear una tabla de mejores resultados en función del nivel de juego.

Adicionalmente a los objetivos propios del proyecto, personalmente me ha enriquecido como informático al introducirme en el mundo del desarrollo de videojuegos profesionales, ya que durante los años de carrera en la universidad, aunque he implementado juegos en asignaturas de Programación e Inteligencia Artificial, ninguno de ellos se ha realizado con una herramienta tan potente como la que he usado para este proyecto y con la posibilidad de tener un interfaz gráfico tan bueno.

Al comenzar el proyecto, uno de los objetivos principales era adquirir información y conocimientos para desarrollar un videojuego, pero además me di cuenta de que no sólo es suficiente con ser un buen programador, sino que además hay que ser un buen diseñador, ya que una de las cosas más importantes de un videojuego, es que tenga una gran calidad audio-visual.

Las características más importantes para la consecución de los objetivos se basan en los siguientes puntos:

- **Conocer en profundidad la API de XNA.**

  Aunque ya tenía experiencia en programar en C#, mi experiencia con *XNA* era nula, por lo que el primer paso fue obtener conceptos y profundizar en los conocimientos y la mecánica de trabajo que requiere la creación de un videojuego.

- **Establecer un análisis de requisitos.**

  El análisis de los requisitos se ha realizado realmente dos veces, ya que hay que establecer los requisitos para la parte visual, y por otro lado los requisitos para la parte meramente de programación.

  Para el análisis de requisitos funcional, hubo que establecer cuestiones como elegir el número de objetos que contendría el juego, el comportamiento de las bolas cuando colisionasen tanto con bolas de su mismo color, como con bolas de diferente color, o incluso cuando no colisionase con bolas, sino con las paredes del tablero. Además había que establecer un sistema de puntuaciones y definir en qué condiciones finalizaría una partida.

Para el análisis de requisitos visuales hubo que determinar anteriormente las opciones básicas que incluiría el juego, empezando por crear un menú donde el jugador pudiese elegir el nivel de dificultad con el que desearía jugar, acceder a las instrucciones del juego, ver la tabla de máximas puntuaciones, etc.

A continuación, hubo que crear los objetos del juego y decidir cómo se dispondrían en la pantalla y, finalmente, elegir el audio para que sincronizado con diversas acciones durante el juego le diese una sensación de mayor logro a la jugabilidad.

- **Desarrollo del proyecto.**

A medida que iba desarrollando el proyecto, han ido surgiendo nuevas ideas tanto por mi parte como por la del tutor que han desembocado en ir renovando los requisitos del juego. Durante el tiempo que ha durado el desarrollo, se han ido estableciendo hitos consensuados con el tutor para ir viendo la evolución del mismo, y aunque la mayor parte de los hitos se han ido cumpliendo, la inclusión de nuevas funcionalidades hicieron forzar en alguna ocasión tener que remodelar el diseño, desembocando finalmente en un juego cuya mayor fortaleza está en la fiabilidad y la jugabilidad.

Los hitos más importantes que se han seguido son los siguientes:

- Aprender las funcionalidades de creación de un juego.
- Conocer la API XNA.
- Análisis de requisitos funcionales y técnicos.
- Diseño de los objetos participantes en el juego.
- Desarrollo del juego. Dentro de este hito, se han realizado varios sub-hitos, entre los que destacan los siguientes:

    - Mover un objeto por pantalla.
    - Detección de colisiones.
    - Algoritmo de eliminación de objetos en las colisiones de objetos múltiples.
    - Evaluación del juego, jugabilidad y fiabilidad.
    - Establecer niveles de jugador y juego.
    - Diseño de los menús y pantallas.
    - Evaluación del juego en su conjunto.

1.4. Contenido de la memoria

La memoria consta de cinco grandes capítulos en los que se profundiza en el tema tratado. A continuación se muestra un breve resumen de cada uno de ellos tratando de manera genérica cada uno de los apartados.

En el capítulo 1, **Introducción**, se hace una breve descripción del tema principal del proyecto, la motivación que me ha llevado a realizarlo y los objetivos cumplidos, tanto los propios del proyecto como los personales. Se hace referencia de manera muy breve a lo que significan los videojuegos tanto para la informática como para los usuarios en la sociedad actual.

En el capítulo 2, **Estado de la Cuestión**, se comenzará por realizar un pequeño repaso a la corta historia del negocio de los videojuegos y qué impulsó a las empresas a dedicar tiempo y dinero a este mercado. Se habla de cómo comenzó todo y por qué, desde los primeros juegos y máquinas recreativas hasta las plataformas más novedosas del mercado y el futuro que les aguarda. Se explicará cómo nace un videojuego profesional desde que comienza la idea hasta que se comercializa en el mercado.

Finalmente se hará mención a la arquitectura XNA elegida para el desarrollo de este proyecto, el motivo por el que se ha elegido y se analizarán qué otras arquitecturas existen en el mercado similares a ésta.

En el capítulo 3, **Gestión del proyecto,** se hace hincapié en las distintas fases que ha atravesado el proyecto, desde la fase de análisis, pasando por la fase de diseño y finalmente por la fase de implementación.

En el capítulo 4, **Objetivos del PFC**, se exponen las conclusiones y reflexiones obtenidas tras la realización de este proyecto, analizando además si se han alcanzado los objetivos marcados en un principio.

En el capítulo 5, **Líneas Futuras**, se hace referencia a las posibles ideas de continuación que han surgido durante la realización del proyecto.

Por último comentar que, al final de este documento, se encuentra un **Anexo** que recoge la **Planificación** (Anexo A), el **Diccionario de Términos y Acrónimos**, que recoge una definición de los términos que pueden resultar ambiguos al lector, pero que a la vez son básicos para la comprensión total del proyecto. Y las **Referencias**, que presenta todas las fuentes que se han consultado para profundizar en los distintos temas expuestos en el proyecto.

Capítulo 2

Estado de la cuestión

En este capítulo se hará un repaso a la historia de los videojuegos, desde que se inventó el primer videojuego hasta los juegos más sofisticados y actuales.

A continuación se mostrará cual es el ciclo de vida de un videojuego, desde la idea original hasta puesta en el mercado.

Finalmente se explicará la arquitectura escogida para el desarrollo del proyecto, la razón de haberla escogido y las arquitecturas rivales.

2.1. Antecedentes

Para poder establecer un punto de inicio, lo primero que hay que saber es qué es un videojuego [3]. Según la Real Academia de la Lengua, un videojuego es "*un dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor o de un ordenador*".

Según esta definición y adaptándola lo más remotamente posible, todo comienza en el siglo XIX, donde en el año 1894 apareció el *Automated Skill Shooter* [4] de la figura 1. Se desconoce su verdadero mecanismo pero sin duda podemos considerarla una de las primeras máquinas de ocio pensadas para un público masivo. Otras máquinas de principios del siglo XX fueron *The Erie Digger, Baffle Ball* (antecesor del pinball) y por supuesto, el mítico *futbolín.*

Todos estos juegos "*prehistóricos*" no se adaptan a la definición de videojuego actual, sin embargo, simbolizan el comienzo de los primeros juegos basados en máquinas.

Figura1. Skill Shooter

2.2. Historia de los videojuegos

En **1947** Thomas T. Goldsmith y Estle Ray Mann patentaron un sistema electrónico de juego que simulaba el lanzamiento de misiles contra un objetivo basado en las pantallas de radar que usaba el ejército americano en la segunda guerra mundial [5] como se ve en la figura 2. El sistema funcionaba con válvulas y usaba una pantalla de rayos catódicos. Permitía ajustar la velocidad y la curva del disparo, pero los objetivos estaban sobre impresionados, no había movimiento de video en la pantalla, pero se ajusta perfectamente a la definición actual de un videojuego, por lo que se podría decir que es el primer videojuego de la historia.

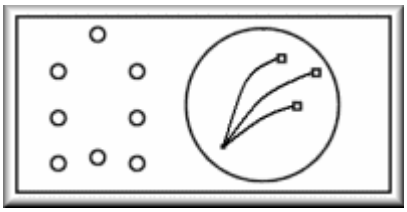

Figura 2. Lanzamiento de misiles

2.2.1. Orígenes

El primer juego gráfico del que hay constancia data del año **1952** en el que el joven Alexander Sandy Douglas presenta su tesis de doctorado en matemáticas en la Universidad de Cambridge (Inglaterra) sobre la interactividad entre seres humanos y computadoras. Se trata de una versión del "*Tres en Raya*" (Tic Tac Toe) para una computadora EDSAC[8], diseñada y construida en esa misma universidad. El programa tomaba las decisiones correctas en cada momento del juego según el movimiento realizado por el jugador, que lo hacía mediante un dial telefónico de rueda que incorporaba la computadora EDSAC. Este juego al carecer de movimiento en la pantalla no se le puede considerar como un videojuego, pero para la época era lo más aproximado.

El origen real del primer videojuego como definición estricta del concepto que conocemos hoy en día, se remonta al año **1958** cuando William Nighinbottham creó un juego llamado Tenis Para Dos (Tennis for two) usando un osciloscopio de laboratorio que consistía en interceptar una bola que cruzaba la pantalla moviendo una línea que hacía de paleta. Su autor lo mostró como curiosidad científica, nunca patentó su invención por lo que no fue hasta catorce años después que una empresa pudo comercializarlo con el nombre de *Pong* y su aspecto era como el que aparece en la figura 3. De esta manera comienza la era de los videojuegos.

Figura 3. Máquina del juego Pong

2.2.2. Evolución de los videojuegos

- Década de los 70

La verdadera explosión de los videojuegos data de la década de los 70 [6], cuando llega al mercado *Computer Space*, una versión del *Space War* representado en la figura 4 y creado por Steve Russell en **1961**. Este juego cuya misión era disparar a los platillos volantes que iban apareciendo en la pantalla, puede decirse que fue el primer juego comercializado con ánimo de lucro, ya que para jugar había que insertar monedas. Los creadores de esta máquina posteriormente fundarían la empresa **Atari**.

Figura 4. Space War

Atari [7] originalmente se llamó **Syzygy**, un término astronómico, pero como ese nombre ya había sido registrado por otra empresa anteriormente, Bushnell, uno de los creadores, eligió la palabra Atari que en japonés significa que una ficha o un grupo de fichas están en peligro de ser capturadas por el oponente. Debido a su significado en japonés, tuvo el éxito que se esperaba.

Debido a la competencia que había en la época, Bushnell creó de manera secreta un "*competidor*" llamado **Kee Games**, que estaba liderado por Joe Keenan. De esta manera, los distribuidores siempre comprarían a alguna de sus empresas, sin saber que las dos pertenecían a la misma persona. Fue tal el éxito obtenido hasta que se descubrió la farsa, que Keenan fue ascendido a presidente de Atari en 1974.

En **1976** aparece *Colossal Cave Adventure*, primer juego de aventuras y una alternativa a los juegos del momento, cuyo potencial era buscar la habilidad y reflejos de los jugadores. Este tipo de juegos se basan en la toma de decisiones, solución de acertijos, y elección de caminos, para llegar a un final bueno o malo. Gracias a este juego se ha evolucionado a las *aventuras gráficas* [9].

En esta década aparecen las primeras videoconsolas, pero no será hasta los años 80 cuando se popularicen.

➢ Década de los 80

Los años 80 comenzaron con un fuerte crecimiento en el sector del videojuego alentado por la popularidad de los salones de máquinas recreativas y de las primeras videoconsolas aparecidas durante la década de los 70. Aunque Atari dominaba casi completamente el mercado de las videoconsolas caseras, experimentó su primera competencia férrea en **1980** con la consola *Intellivision* de **Mattel** [8], que hizo publicidad de su mayor capacidad gráfica, comparada con la

Atari 2600. A pesar de esto, la 2600 siguió siendo el estándar de la industria dada su superioridad en el mercado, y además el número de títulos disponibles para su consola.

Sin embargo, Atari tuvo problemas a comienzos de los 80. La división de ordenadores personales (PC's), consolas de videojuegos y máquinas de *arcade*[10], operaban independientemente dentro de la empresa y rara vez cooperaban entre sí.

En los primeros años de esta década, el negocio asociado a la industria del videojuego alcanzó grandes cosas en estos primeros años de los 80, pero sin embargo, en **1983** comenzó la llamada crisis del videojuego, afectando principalmente a Estados Unidos y Canadá, y que no llegaría a su fin hasta **1985** debido a la proliferación de los ordenadores personales, puesto que los jugadores preferían piratear o comprar los juegos que había en las recreativas, en vez de gastarse el dinero en ella. La solución al problema fue la creación por parte de las empresas de sus propias consolas caseras, o desarrollar software para éstas. De esta manera nace la consola de la figura 5, la *Nintendo Entertainment System (NES)*, de **Nintendo** [9].

Otras empresas como **Sega** [10] copiaron la idea y ésta creó la consola de la figura 6, la *Master System*, **Commodore** la *Amiga* y **Atari** la *7800* de la figura 7, siendo las 2 primeras las que dominan en el mercado en España. A finales de los 80 se introdujeron las consolas de 16 bits, como la *Mega Drive* de Sega.

Figura 5. Nintendo Entertainment System

Figura 6. Master System

Figura 7. Atari 7800

A finales de los 80 Atari en un último intento y Sega adelantándose una vez más a sus competidores revolucionaron y reactivaron el mercado. Sega lanzó en **1989** la *MegaDrive* [11], una consola compatible con los juegos de la *Master System*, pero mucho más avanzada, ya que doblaba la potencia de la NES y su anterior aparato, su revolucionario aspecto aparece en la figura 8. La NES siguió rivalizando con MegaDrive, pero esto permitió a Sega llegar a dominar el mercado hasta **1992**, fecha en la que *SuperNintendo* llega a Europa y frena la escalada de Sega.

Figura 8. MegaDrive

La alternativa a las consolas estaba por tanto en los ordenadores personales, como el *Sinclair ZX Spectrum* [12] de la figura 9, que era un pequeño ordenador de 8 bits basado en el microprocesador Zilog Z80A, fabricado por la compañía británica **Sinclair Research** y lanzado al mercado el 23 de abril de **1982**.

En Europa, el Sinclair ZX Spectrum fue uno de los microordenadores domésticos más populares de los años 80. Su optimizado y compacto diseño hizo las delicias de miles de aficionados a la informática y los videojuegos. Aún hoy perduran miles de fans del Spectrum que siguen jugando a sus juegos con emuladores. Además hay un mercado de coleccionismo tanto de cintas de juegos originales como de los propios Spectrum.

Figura 9. Sinclair ZX Spectrum

Por otro lado, **Amstrad** comenzó a comercializar sus propios ordenadores personales en un intento por capturar el mercado de *Commodore* y *Sinclair* con el *CPC 464* [13] de la figura 10 en **1984**. El ordenador, basado en un microprocesador Zilog Z80A de 8 bits a 3,7MHz y con 64 Kbytes de memoria RAM incluía además la unidad de cinta y un monitor (de fósforo verde o color) con la

fuente de alimentación integrada. La gama CPC se lanzó en Francia, Reino Unido, Alemania, Australia y España, donde fue un éxito de ventas. No logró desbancar a los competidores, pero en adelante todo videojuego para ordenador doméstico que tuviese aspiraciones de éxito de ventas se versionará para esos tres sistemas.

Figura 10. Amstrad CPC 464

Durante estos años, en pleno apogeo de los salones recreativos además de los ordenadores personales, aparecieron tal vez los juegos más adictivos de la historia, los denominados juegos arcade, ya que a día de hoy son muchos los millones de personas que siguen jugando a estos clásicos.

En **1981**, Nintendo lanzó al merado *Donkey Kong* [14]. Este juego se basaba en mover a un pequeño personaje por unas plataformas con escaleras para rescatar a su novia, prisionera de un gorila que lanzaba objetos desde lo alto. Este sencillo juego tuvo muchísimo éxito, tal es así que se han hecho multitud de versiones de este título. Se puede apreciar la pantalla del juego en la figura 11.

Figura 11. Pantalla del Donkey Kong

En este mismo año, Atari presenta *Haunted House* [15]. En esta aventura que se muestra en la figura 12, el jugador era representado como un par de ojos y debía recorrer una vieja mansión con el objeto de encontrar las piezas de una urna, evitando a las diversas criaturas que habitaban allí, para después volver al principio para abandonar el edificio. A partir del segundo nivel, todo estará a oscuras teniendo que buscar las puertas y paredes al tacto, recreando así el miedo.

Figura 12. Pantalla del Haunted House

En **1982**, en plena crisis aparecen tal vez los títulos más exitosos, como *Battle Zone, Zaxxon* o el más conocido por todos, el *Pacman*, del cual se puede ver la pantalla del comienzo del juego en la figura 13.

Figura 13. Pantalla inicial del Pacman

El año del cine en los videojuegos fue **1983**. Por primera vez se incluía el argumento de una película en un videojuego como fue *Tron*. A partir de aquí empezaron a salir más juegos basados en películas como *Star Wars* de *Atari* o la segunda parte de *Tron*.

Otro juego conocido que aparece este año es el *Dragon's Lair*, del cual podemos ver una fase del juego en la figura 14, y que utilizaba por primera vez la tecnología Laser Disc. Por otro

lado, Nintendo lanza el *Donkey Kong 3,* que cambia el estilo de sus antecesores, y por esta causa no llegó a triunfar.

Figura 14. Fase del juego Dragon's Lair

Sin embargo, lo más destacable de este año fue el lanzamiento de uno de los títulos con más éxito de todos los tiempo, *Mario Bros* [16]. Este personaje ya apareció en el Donkey Kong, pero ahora tenía su propio juego. Se dice que le llamaron Mario por el gran parecido con el casero del diseñador del juego, Miyamoto, en el edificio donde se hospedaba en Nueva York cuando tenía que ir allí. También aparece por primera vez su hermano Luigi.

El objetivo del juego es derrotar a todos los enemigos en cada nivel. Los dos extremos de cada nivel tienen una característica mecánica que le permite al jugador salir por la izquierda y aparecer a la derecha, y viceversa. Cuantos más niveles cruce Mario, mayor será la dificultad, y aparecerán un número mayor de enemigos cada vez más fuertes y muchísimo más difíciles, como se aprecia en la figura 15.

Figura 15. Pantalla del juego Mario Bros

España no era un país que destacase por crear ni juegos ni consolas, sin embargo, en 1983, la primera compañía que comenzaría su andanza distribuyendo los primeros ZX Spectrum y Amstrad CPC en nuestro país fue **Indescomp**, pero que también sería la primera en distribuir videojuegos de creación española, como fue *La pulga* que curiosamente tuvo más éxito en Inglaterra.

A finales de este año sale *Final Fantasy*, otro exitazo en ventas que a día de hoy ya va por la decimocuarta edición. El lema del juego es "*Cuando la oscuridad amenace con destruir el planeta, cuatro guerreros escogidos saldrán de las sombras, cada uno con un cristal con el fin de restablecer el equilibrio entre la luz y la oscuridad*". En la figura 16 se muestra la portada del juego para PlayStation.

Figura 16. Carátula del juego Final Fantasy

Por último, hay que destacar la aparición de una empresa nueva que nació en Estados Unidos llamada *Electronic Arts (EA)* [17]. Nació como una productora de videojuegos, y se decidió especializar en este sector con juegos apropiados para el soporte, como los géneros de estrategias y militar, sin embargo, el éxito de EA llegaría una década después, por lo que no nos extenderemos más de momento.

Mientras en el resto del mundo los videojuegos sufrían una crisis, el año **1984** fue un hito dentro de la historia del videojuego en España, ya que fue el año en que aparecieron las primeras empresas españolas dedicadas al software de entretenimiento, convirtiendo este año en el primero de lo que se ha llegado a conocer como la "Edad de Oro del Videojuego Español" [18]. Durante este año, Indescomp publica su segundo juego, "*Fred*", aparece **Dinamic Software** (Dinamic Multimedia a partir de 1993) con juegos como *Yenght* y *Artist*, y también **Erbe**, que se dedicará principalmente a la distribución de videojuegos durante esta década.

En noviembre de 1984 también nace **MicroHobby**, una de las primeras revistas españolas dedicada a los videojuegos, aunque solamente a la plataforma Spectrum en sus inicios. En la figura 17 aparece una de las primeras portadas de la revista.

Figura 17. Portada de la revista MicroHobby

Durante este año sale a la venta la que se considera la primera aventura gráfica de la historia. *King Quest I* [19], desarrollada por **Sierra Online** y diseñada por Roberta Williams, junto a otros seis programadores. Salió para IBM PCJr, Master System, PC, Atari ST y Amiga. El juego fue toda una revolución y detrás de su desarrollo hay un trabajo de más de 18 meses y un presupuesto de más de setecientos mil dólares. Han llegado a sacar hasta 8 partes de este juego (la última en 1998). El *King Quest VIII. The mask of Eternity* fue el único de la saga que poseía gráficos en 3D, cosa que provocó críticas de una gran parte de seguidores del juego. La siguiente figura muestra los gráficos del juego.

Figura 18. Fase del castillo del King's Quest I

Durante este año, sale al mercado *Super Mario Bros*. *Mario Bros* y Luigi deben rescatar a la princesa Peach, del Reino Champiñón, que fue secuestrada por el rey de los koopas, Booser. A través de ocho diferentes niveles de juego, los jugadores pueden controlar a alguno de los dos hermanos y

deben enfrentarse finalmente a cada uno de los monstruos de cada castillo para liberar a Peach. A continuación se muestra en la siguiente figura a Mario en una de las pantallas del juego.

Figura 19. Super Mario Bros

Otro de los juegos más importantes de la historia fue el mítico *Tetris* [20] inventado por Alexey en **1985**. Su primer juego incluyó siete formas construidas a partir de bloques. Tetris empezó a hacerse famoso más adelante cuando Gerasimov lo programó para ordenadores personales. Más tarde se programó para Apple II y Commodore 64. La figura 20 muestra una de las muchas variedades gráficas del juego.

Figura 20. Pantalla de Tetris

En los salones recreativos, los juegos más demandados en **1986** eran el *Out Run*, *Bubble Bobble* y *Arkanoid*. Durante esta época, los adolescentes pasaban horas tratando de rescatar a las novias de Bub y Bob o intentando conducir el Ferrari Testarossa lo más lejos posible antes de que se le acabase el tiempo. La figura 21 representa el inicio del juego.

Figura 21. Pantalla inicial del Out Run

Pero realmente lo que marcó este año fue la salida al mercado por parte de Nintendo del juego *La leyenda de Zelda*. La historia se ambienta en la tierra de Hyrule donde un joven llamado Link tiene que rescatar a la princesa Zelda de las garras de Ganon, el rey del mal.

El videojuego tenía varias innovaciones técnicas. La más destacada era la posibilidad de guardar los progresos aunque apagaras la máquina. Tuvo muchísima aceptación y llegó a vender 6 millones y medio de copias.

A mediados de los 80 la gente ya se había casi olvidado del *Breakout* de Atari. **Taito** aprovecha esta situación y lanza una versión renovada del juego, el revulsivo *Arkanoid* [21] de la figura 22.

La "*pala*" se sustituye por una nave llamada Vaus. El juego contaba con 33 fases y al final nos enfrentábamos al enemigo final llamado Doh. La jugabilidad fue revisada y ahora nos encontrábamos con que los ladrillos eran de varios tipos, que algunos soltaban cápsulas para dar mejoras a nuestra nave, se aumentó el número de bolas, etc. Estaba a años luz de Breakout y se acabó convirtiendo a casi todos los sistemas de la época.

Figura 22. Pantalla del juego Arkanoid

Si por algo se recordará el año **1987** es por la calidad de los juegos de tres compañías como son **Sega**, **Capcom** y **Taito**, las reinas de los juegos denominados Arcade.

Sega lanzó éxitos como *After Burner*, *Shinobi* y *Wonder Boy*. Por otro lado, Capcom maravilló con 1943 y *Street Fighter*, mientras que Taito triunfaba en los salones de videojuegos con *Operation Wolf*.

En este año aparece la primera versión de *Metal Gear*. **Konami** lo publicó para NES y su éxito radicó en que por primera vez, un juego de guerra no se basaba en acribillar todo lo que aparecía por delante, sino que había que aplicar inteligencia e ir recabando armas hasta conseguir la Metal Gear. La figura 23 muestra una de las fases del juego.

Figura 23. Fase del juego Metal Gear

En **1988** llegó al mercado uno de los mejores juegos de creación española de mano de **Opera Soft**, al menos hasta la época, se trata de *La Abadía del crimen* [22]. Con una alta dificultad, el juego nos trasladaba a un monasterio en donde debíamos resolver un crimen manejando al monje franciscano, Guillermo de Occam, que durante 7 días deberá investigar a la vez que cumple con las tareas típicas de la abadía. Este juego tuvo mucho éxito en los ordenadores Amstrad CPC y Spectrum.

En la figura 24 se aprecia que la calidad de los gráficos no era muy buena, pero lo compensaba con la originalidad.

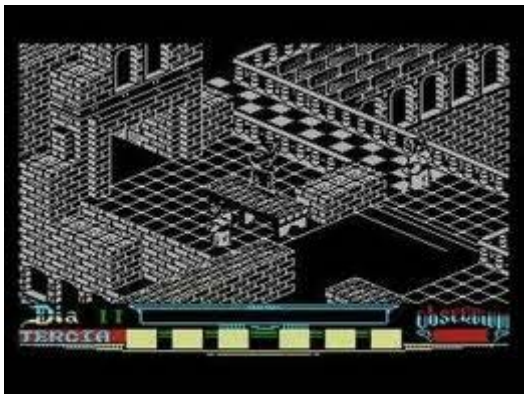

Figura 24. La Abadía del crimen

En **1989** se produjo una revolución con *Sim city* [23]. Su creador, Will Wright, un fanático de los videojuegos pensó que era más divertido crear juegos que simplemente jugar, por lo que inventó una manera distinta de divertir al usuario creando su propio juego. Básicamente se trata de dotar al jugador del poder de ser alcalde de una ciudad e intentar hacerla crecer creando industrias, colegios, zonas residenciales, etc. La versatilidad del juego estaba en que se parecía mucho a la realidad, y había que controlar factores como la delincuencia, la seguridad ciudadana, etc. Este juego tuvo un tremendo éxito y supuso la creación de un subgénero dentro de la estrategia. En la figura 25 se muestra cómo se puede ir creando una ciudad.

Figura 25. Juego Sim city

Por último, llegó al mercado uno de los juegos arcade más carismáticos aún hoy, *Pang.* En el juego cada fase empieza con un cierto número de globos de distintos tamaños, que al tocar al personaje le quitan una vida. Tendremos que usar nuestra arma para disparar a las bolas y hacer que se dividan en otros trozos más pequeños y así hasta que desaparezcan. La figura 26 muestra al jugador y las bolas de diferentes tamaños que debe eliminar.

Figura 26. Pang

➢ Década de los 90

Los años 90 destacan principalmente por la evolución de los juegos. Durante las décadas anteriores realmente hubo saltos tecnológicos, pero en esta década, se podría decir que cada año se avanzaban 10 años de las décadas anteriores.

Durante este tiempo aparecen las mayores rivalidades entre las empresas no sólo por los juegos, sino por las videoconsolas, ya que la que tuviese la consola con más aceptación se ganaría prácticamente todo el mercado. Entre las consolas que más éxito tuvieron fueron las siguientes:

En **1990**, Nintendo lanza al mundo la SNES de la figura 27, consola que tendrá una de las máximas rivalidades, que han existido en la historia de los videojuegos, con la Mega Drive de Sega.

Figura 27. SNES

También aparece la *Game Gear* por parte de Sega para hacer competencia a la Game Boy de Nintendo en cuanto a las consolas portátiles. Ambas en la figura 28.

Figura 28. Game Gear y Game Boy

Este mismo año apareció *Neo Geo* [24], una consola muy adelantada a su época ya que permitía a los jugadores tener las mismas experiencias que en una máquina recreativa, pero en casa. El mayor problema fue su elevado precio y el de los juegos, por lo que podría decirse que **SNK**, la empresa que lo lanzó, era la Rolls Royce del mercado de los videojuegos. En la figura 29 se muestra la Neo Geo para hogar.

Figura 29. Neo Geo

Amstrad trató de entrar en el mercado de las videoconsolas con la *GX4000*. Contaba con un procesador de 8 bits a 4 Mhz, con 64 KB de RAM y 32 KB de ROM.

La consola tenía buenos gráficos y una gran paleta de 4096 colores superando a la Mega Drive. Su catálogo contó con tan sólo 40 juegos, de los cuales la mitad eran exclusivos de la consola, pero sin embargo, la máquina fue un fracaso comercial, siendo poco popular porque empleaba tecnología 8 bits, frente a los 16 bits de la Super Nintendo y la Mega Drive. También tuvo muy pocos juegos en su lanzamiento y muchos otros fueron acabados meses más tarde o cancelados.

En **1991**, Sega veía que estaba perdiendo terreno respecto a Nintendo, por lo que inventó el *Mega CD*. Aprovecharon las capacidades de los CD-ROM permitiendo a los programadores crear juegos más largos, con mejor música y secuencias de video digital. La idea fue buena, pero no tuvo mucho apoyo popular por lo que unido a que no sacaron muchos más juegos para esta plataforma, poco más tarde acabó por fracasar.

En **1994** se vivió una de las peleas más fuerte por la supremacía del mercado. Sega entró fuerte con la *Saturn* [25]. Tenía una CPU de 2 x 32 bits (28,6 y 20 Mhz) con 2Mb de RAM y una unidad de CD 2x.

Pero la llegada de una nueva consola de Sony (la PlayStation) era inminente y Sega tenía que hacer algo para no verse superada por la competencia. Aunque en lugar de solucionar las cosas, se las complicó aún más. Rediseñó la consola con prisas con el resultado de que hubo muy pocos juegos en su lanzamiento y los que estaban listos para salir a la venta con ella habían sido acabados con prisas.

Saturn era una plataforma ideal para los juegos con gráficos en 2D, muchos de ellos imposibles de ver en la consola de Sony. Se promocionó muy bien en Japón y Estados Unidos y todo hacía ver que la consola se vendería bien, pero lo que Sega no se esperaba es que la *PlayStation* se fuera a convertir en una de las consolas más vendidas de la historia, afectando seriamente a las ventas de su sistema.

Con la salida de la consola de Sony al mercado, la Saturn de la figura 30, fue perdiendo terreno y más adelante, con la llegada de la Nintendo 64 quedó relegada al tercer lugar por la lucha en el mercado de los videojuegos.

Figura 30. Saturn

Casi simultáneamente a Saturn, una nueva empresa irrumpe con fuerza en el mercado, se trata de **Sony** y su poderosa PlayStation.

La consola fue concebida después de la ruptura del acuerdo de Sony con Nintendo. El acuerdo consistía en que Sony desarrollaría el CD-ROM para la nueva SNES PlayStation (Snes CD Rom) de Nintendo, con el cual se pretendía dotar de mayores capacidades multimedia a la nueva consola de Nintendo, de la misma forma que Sony colaboró anteriormente en el desarrollo de SNES, fabricando su chip de sonido, que permitía 8 canales simultáneos con calidad *PCM*[11], quedando muy por encima de la competencia. Sin embargo, Nintendo también encargo ese proyecto a Philips, lo que

provocó que, al descubrir Sony el juego a dos bandas de Nintendo, empezara a fabricar juegos para la competencia (como Hook para Sega Mega-CD). Esto y el poco éxito de las consolas que en aquel momento incorporaron el CD, provocó que Nintendo cancelara el proyecto. A Sony no le pareció acertada la determinación de Nintendo y, gracias a la experiencia adquirida, lanzó por su cuenta su propia consola, con la cual ha obtenido un éxito sin precedentes y se puede ver en la siguiente figura.

Figura 31. PlayStation

Nintendo por su parte, intentó contraatacar un poco más adelante con la consola de la figura 32, la *Nintendo 64* en **1996**. Se trataba de una consola de 64 bits con muchas innovaciones en la época, como por ejemplo el mando de control (el primero en llevar unos botones dispuestos en forma de cruz) o la vibración de los mandos. Sin embargo, el soporte de almacenamiento de la consola tenía limitaciones que perjudicaba a la competitividad del mercado. Una limitación importante fue la poca capacidad y el alto coste de producción de los cartuchos en lugar de los CD-ROM, formato utilizado por los competidores. La limitada capacidad de los diseñadores de juegos, obligados a luchar con el contenido del juego para encajarlo en un espacio limitado, a pesar de que el tiempo de acceso es menor en un cartucho, era un factor demasiado fuerte para salvar. Otro inconveniente era una técnica limitada de texturas caché, ya que sólo podía contener texturas de pequeñas dimensiones y profundidad y de colores reducidos, que tuvo que ser estirado para cubrir grandes superficies en el juego. Por todo ello, la Nintendo aunque luchó por ganar la batalla a la PlayStation, no lo consiguió.

Figura 32. Nintendo 64

Lo que no se le podrá recriminar jamás a Sega es que no ha intentado estar a la altura, y de hecho el último coletazo lo dio en **1999** con el lanzamiento de *DreamCast*, cuyo diseño se ve en la figura 33. Contaba con una CPU a 200 Mhz y utilizaba como soporte el GD-ROM. Fue la primera consola de 128 bits que salió al mercado.

Venía marcada por el fracaso de Saturn y la competencia sería muy dura con PlayStation 2 (que venía del exitazo de su antecesora). Hubo muchos factores y hay muchas hipótesis explicando que pasó para que la *DreamCast* [26] fracasara de la manera que lo hizo, pero sobre todo las luchas internas dentro de Sega y las pérdidas económicas influyeron en la retirada del mercado.

Lo cierto es que muchos, años más tarde, se la considera como una de las mejores consolas que han existido.

Figura 33. Dreamcast

Y no hay que olvidarse de los juegos. Aparecen joyas como *Monkey Island*, *Doom*, *Street Fighter II*, *Lemmings* y el famosísimo *Sonic*, de los cuales se hablará a continuación.

La década comenzó con juegos basados en consolas de 16 bits, pero rápidamente llegaron las de 32 bits y sin apenas ponerse a la venta ya se hablaban de los 64 bits. Los usuarios empezaban a demandar cosas nuevas, y aparecieron los primeros juegos en 3D como los prestigiosos *Doom* y *Alone*

in the Dark. A finales de los 90 aparecieron consolas de 128 bits, y juegos como *Tomb Raider*, *Resident Evil* y el *Gran Turismo* triunfaron sobre todos los demás por su alta calidad tanto en originalidad, jugabilidad y sobre todo en gráficos y sonidos, algo que el jugador siempre valora a la hora de adquirir un juego.

Si por algo se caracteriza el comienzo de los 90 en cuanto a videojuegos fue la guerra delarada entre Nintendo y Sega con sus Super Mario Bros y Sonic.

Cuando Super Mario arrasó en el mercado, Sega se quedó rezagada y tuvo que ponerse las pilas, por lo que en **1991** nació *Sonic* [27], un divertido erizo azul cuyo objetivo era salvar a sus amigos animales secuestrados por el Doctor Robotnik, que quiere utilizarlos como fuente de energía para su ejército de robots con la intención de conquistar el planeta Mobius.

Para conseguir su objetivo, Sonic tendrá que recolectar las 7 Esmeraldas del Caos. Se puede aprecia una escena del juego en la figura 34.

Figura 34.Escena de Sonic.

La aparición de Sonic fue un gran éxito mundial y el juego llegaría a los 4 millones de copias vendidas. En Estados Unidos fue elegido como el personaje más querido por delante de Micky Mouse y Michael Jordan. Tal fue el éxito del juego que durante años y aún hoy en día siguen saliendo secuelas con más aventuras de este atractivo y divertido personaje.

Este mismo año cabe destacar el lanzamiento de los Lemmings, del cual se sacaron varias versiones debido a su tremendo éxito y es uno de los juegos que a más diferentes plataformas se ha portado. Se trataba de dirigir las acciones de un grupo de pequeños seres para llevarlos a la salida de cada nivel.

En cuanto a las máquinas recreativas, se puede decir que el gran percusor de este modelo fue *Street Fighter II*. La figura 35 muestra una de las escenas más famosas del juego.

Figura 35. Escena de Street Fighter II

El año **1992** se caracterizó por ser un año donde aparecieron éxitos como la segunda parte de Sonic, que con su nuevo compañero Tails, dotó al juego de más dinamismo, más velocidad y más duración.

Tras el éxito de Street Fighter II, muchas empresas decidieron invertir en juegos de lucha, uno de los que triunfaron y fue competencia directa de Street Fighter II fue *Mortal Kombat* [28]. El juego constaba de siete luchadores y nuestra misión era conseguir ser el campeón del torneo "*Mortal Kombat*". Había una gran variedad de golpes y el juego era tremendamente violento, pudiendo realizar un "*fatality*" (para matar al rival), que resultaba muy violento. Para el diseño de los personajes se utilizaron actores reales.

Este juego fue muy criticado por su violencia y en algunos sitios se llegó a prohibir. Pero esto en lugar de afectarlo negativamente, lo que hizo es que aumentaran sus ventas y popularidad. A continuación en la siguiente figura se puede apreciar lo tremendo del momento citado.

Figura 36. Acción fatality de Mortal Kombat

Como se ha visto hasta ahora, lo más predominante en la época eran los juegos donde lo que más importaba era la sencillez del argumento y en mayor o menor medida, la violencia, por lo que aprovechando el tirón, pero profundizando en la calidad gráfica apareció el revolucionario *Alone in the Dark* [29], en el cual el jugador encarna a un detective que es contratado para realizar un inventario de los objetos que se encuentran en una mansión, cuyo dueño se ha suicidado hace poco. La mansión resulta estar llena de zombies y criaturas horrorosas, por lo que el objetivo para por sobrevivir y huir de la mansión. En este juego se impuso por primera vez el entorno 3D lo que dotaba al jugador de un espacio visual desconocido hasta el momento, y eso unido a la originalidad del argumento y la durabilidad del juego, le proporcionó un éxito enorme. En la figura 37 se ve cómo aparece el diseño 3D por primera vez.

Figura 37. Escena 3D de Alone in the Dark

La plataforma de los PC's vivieron su esplendor en **1993** con la aparición de *Doom* [30]. El protagonista era un marine de los Estados Unidos que, después de golpear a un superior por ordenar disparar contra civiles es mandado a una base de Marte donde se desenvuelve toda la acción. El objetivo de cada nivel es ir encontrando el interruptor de salida del sector correspondiente. El juego se puede decir que era algo tosco para lo avanzados que estaban ya los juegos, pero tenía muy buenos modelados y un gran mapa para perderse entre almacenes, puertas y paredes.

Además de los éxitos ya citados, cabe destacar la explosión de la empresa EA, de la cual ya habíamos hablado anteriormente. A principios de los 90 empezó a producir simuladores deportivos de forma masiva y decidió producir para consola, siendo Sega Mega Drive y Super Nintendo, las plataformas en donde la compañía sacó más juegos a mediados durante esta década, y fue en este año cuando sacó al mercado el primer FIFA de todos, el *FIFA 94* [31], un simulador de fútbol donde el jugador podía escoger entre todos los equipos de las principales ligas del mundo, y emular a su ídolo haciendo sus jugadas más destacadas, aunque en esta primera versión los jugadores no tenían los nombres reales. Además también tenía modos de entrenamiento o partido de exhibición.

La nueva generación de FIFA cambió mucho, la inteligencia artificial (IA) de los jugadores siendo más desarrollada. Por ejemplo, en el mano a mano los porteros eran más rápidos y si no te decidías a dar un pase o a meter gol, el portero te podría quitar el balón de los pies. Los guardametas también podían ser expulsados y por primera vez se podían manejar manualmente, lo que dotaba al juego de más dificultad que los anteriores. A continuación se muestra una imagen de este éxito mundial en la figura 38.

Figura 38. Imagen del FIFA 94

Durante tres años se vivió una guerra fratricida entre las empresas punteras por tener el mejor soporte, y se olvidó lo principal, los juegos; sin embargo, en **1996** Capcom supo esperar su momento y aprovechando el bajón del momento respecto a nuevos títulos y sabiendo del éxito que tuvo *Alone in the Dark*, decidió invertir esfuerzo en producir un juego similar pero mucho más atractivo en cuanto a gráficos y argumento, y nació el exitoso *Resident Evil* [32].

La historia tiene lugar en 1998 en la ciudad de Raccoon, donde una organización multinacional llamada *Corporación Umbrella* se muestra como un conglomerado que posee ramas de investigación, desarrollo y comercio de todo bien y servicio que exista, pero que en realidad es la cubierta para una industria de desarrollo y venta de armas (especialmente biológicas) de última tecnología.

La serie comienza con un escape del virus T en la mansión Spencer, que se encuentra a las afueras de Racoon. En esta mansión los trabajadores de Umbrella se convierten en zombis y los STARS deben sobrevivir. Además de los zombis, los protagonistas deben enfrentarse a diversas criaturas o mutaciones creadas por Umbrella y a los distintos virus.

Las entregas comienzan durante o después de la gran oleada destructiva inicial. En esta situación el protagonista, usualmente un miembro del equipo STARS, debe sobrevivir y buscar la forma de escapar mientras se enfrenta a los zombis y monstruos no solo nacidos del contacto con los

muertos, sino también de otros que se han fugado de los laboratorios de desarrollo de armas biológicas de Umbrella.

El contexto bajo el que se desarrolla la historia solo se vislumbra a medida que avanza el juego; los diferentes personajes y lugares que visita el jugador proporcionan pistas respecto a lo que sucedió a la gente y lo que debe hacer. En la figura 39 se muestra una imagen del juego.

Figura 39. Escena de Resident Evil

Ha sido tal el éxito generado que no sólo se han publicado multitud de secuelas del videojuego, sino libros, cómics e incluso la industria del cine en 2002 presentó la primera película basada en este videojuego con un presupuesto de 33 millones de dólares, y recaudó casi 103 millones de dólares. Debido al gran éxito del videojuego, se han seguido filmando más secuelas con éxitos similares.

Durante **1997** los títulos más relevantes fueron *Age of Empires* y *Grand Theft Auto (GTA);* a pesar de pertenecer a géneros totalmente distintos, ambos tuvieron un gran número de seguidores por las novedades que proporcionaron y a continuación surgieron secuelas con tanto o mayor éxito que las originales.

Age of Empires pertenece al género de los juegos de estrategia en tiempo real y utilizaba una vista isométrica para representar los escenarios. Se podía jugar desde la edad de piedad hasta la de Hierro con la posibilidad de elegir numerosas civilizaciones de las respectivas épocas. Dentro del mundo de juegos de estrategia ha tenido un gran éxito y es de las sagas más famosas.

Grand Theft Auto por su parte obtuvo éxito dentro del género de los juegos de acción debido a la libertad de acciones que podía tomar el jugador, aparte de la polémica que creó por su alto contenido en violencia (violencia realista, lo cual alertó a la sociedad estadounidense). Tenía una vista

cenital y un potencial limitado, pero sentó las bases de la saga. En las siguientes figuras se muestran una imagen de estos dos títulos.

Figura 40. Escena de Age of Empires

Figura 41. Escena de Grand Theft Auto

Debido al gran reclamo de los jugadores por GTA, el mercado vio otra oportunidad y **Polyphony Digital** no desaprovechó la ocasión y el **1998** contraatacó con *Gran Turismo.* Era de los primeros juegos que nos introducía en el cuerpo de un conductor novel que tenía que ir sacando licencias para conseguir coches cada vez mejores y potentes. A todo esto se lo sumaban unos grandes gráficos para la época como se aprecia en la figura 42.

Figura 42. Imagen de Gran Turismo

Otro título de gran éxito fue el lanzamiento de *Metal Gear Solid* [33] desarrollado por **Konami**. Actualmente es considerado como el mejor juego de la historia y se especula con la adaptación del videojuego al cine para el año 2011. El primer Metal Gear apareció originariamente en 1987 como hemos visto anteriormente, pero poco después Konami publicó una versión para NES de Nintendo. Esta encarnación de 8 bits de Metal Gear había sufrido algunos cambios secundarios; muchos fondos de pantalla habían cambiado, aunque los mapas sólo habían sido sutilmente alterados.

Metal Gear presentó la idea del suspense a un público de aficionados a los videojuegos que se había acostumbrado a simplemente arrasar todo lo que estuviera a la vista. Fiel a las verdaderas operaciones secretas, el mayor aliado de Solid Snake, el protagonista del juego, no era un rifle o una granada (aunque su importancia nunca se puede subestimar), sino su propia agilidad e inventiva.

En combinación con la complejidad y la profundidad del argumento, estos factores de juego angustiosos contribuyeron a hacer de Metal Gear uno de los juegos de acción-aventura más originales del momento, convirtiéndolo en un clásico de manera instantánea.

El Metal Gear original se juega de forma muy similar a versiones más recientes; la serie es cada vez más elaborada e impresionante, pero la base sigue siendo la misma. La acción se muestra desde una perspectiva aérea, por lo que se pueden distinguir con facilidad los diseños del suelo y a cualquier soldado de guardia.

Durante **1998** podemos destacar dos títulos por encima de todos en el género de ciencia ficción como son el *Half Life* y *StarCraft.*

El primero fue diseñado por **Valve Corporation** y su potencial radica en que los protagonistas jamás son vistos en tercera persona, excepto cuando se muestra una fotografía o se ve la imagen del logotipo que tiene el juego. Tampoco se les oye hablar, sin embargo algunas veces los

personajes que te vas encontrando aluden a respuestas "*imaginarias*" o "*insonoras*" de los protagonistas, lo que da a entender que el protagonista habla aunque no es perceptible desde el punto de vista de primera persona. Estas características proporcionan al jugador una sensación muy realista.

Half Life no tuvo mucho éxito en el momento de su lanzamiento, pero después, al ser aclamado por críticos por la trama del juego, por la sofisticación de la inteligencia aplicada y a su larga duración, se convirtió en un súper ventas, proclamándose juego del año por más de 50 publicaciones lo que lo convirtió en uno de los mejores juegos del género, llegando a ser una saga de culto por miles de fans. En la figura 43 se muestra una escena del juego donde se aprecia lo descrito.

Figura 43. Escena de tiroteo en Half Life

Por otro lado, StarCraft [34] es un juego de estrategia en tiempo real creado por **Blizzard Entertainment** y hasta el día de hoy ha vendido casi 10 millones de copias en todo el mundo, por lo que se ha convertido en uno de los juegos más vendidos para PC, aunque también está disponible en otras plataformas. El juego cuenta con varias secuencias cinematográficas y además, el fabricante facilitó un servidor gratuito, para que los jugadores pudieran competir en el modo **multijugador**[12] a través de Internet, todo un éxito en el momento. En la siguiente figura se muestra una secuencia del juego.

Figura 44. Escena de StarCraft

No hay que olvidarse de uno de los videojuegos españoles más populares hasta el momento desarrollado por **Pyro Studios**, el exitoso *Commandos* [35]. El objeto del juego es dirigir un grupo de comandos militares a lo largo de varios escenarios en un ambiente de la Segunda Guerra Mundial. La vista del juego es de modo aéreo permitiendo al jugador visualizar la totalidad del campo al mismo tiempo. El juego fue el primero de su estilo (infiltración por medio de varias unidades en una vista aérea) por lo cual creó un estilo de videojuegos y desde entonces varios juegos que salen se rotulan como "*Estilo Commandos*". Además es considerado el mejor juego de táctica de la historia. La figura 45 muestra la carátula del juego.

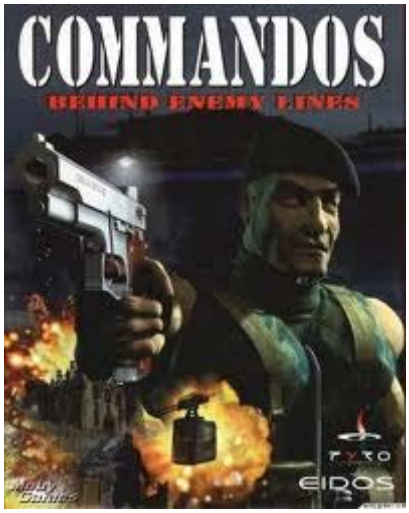

Figura 45. Portada de Commandos

➢ Nuevo milenio

Llegó el año 2000 y pasó la crisis del fin del mundo. Durante los primeros años el mercado se preocupó más de lanzar nuevos soportes para tener la tecnología más avanzada del momento que en hacer juegos de calidad. No fue hasta 2004 cuando sale al mercado *World of Craft* y desata la euforia en el mundo entero con más de 11,5 millones de suscripciones en el mundo a día de hoy. Este juego revoluciona el mundo del videojuego en el sentido de que para poder jugar hay que tener una suscripción la cual hay que pagar. Destaca la originalidad del juego, el amplio abanico de posibilidades para elegir jugadores, roles, entornos, etc. Este juego basado en la técnica de multijugador marca un antes y un después en este tipo de juegos en línea. Más adelante se hablará de él.

Como se ha dicho, durante esta década fueron muchas las videoconsolas que salieron al mercado, y la verdad, las empresas se pusieron las pilas porque fue éxito tras éxito sobre todo durante los primeros años del nuevo siglo.

En el año **2000**, Sony se desquitó del fracaso de Dreamcast y tras el éxito de PlayStation, lanzó al mercado *PlayStation 2 (PS2.* El gran aliciente de esta videoconsola era el lector de DVD ya que en la época los lectores de DVD eran muy caros, y hacerse con una *PS2* era más económico; además de su desmesurada potencia, ofrecía compatibilidad con juegos de su predecesora, por lo que su éxito estuvo asegurado, siendo la consola que más unidades vendió en su momento de salida, y la más vendida de la historia. Poco después salió una versión más pequeña, casi se podría decir que portátil que tuvo un gran reclamo por parte de los jugadores. La figura 46 muestra la versión *delgada* (del inglés *slim)* de esta joya.

Figura 46. PlayStation 2 y mando

En **2001**, el gigante de la informática, **Microsoft**, en unión con Sega, y tras unos años pensando en cómo entrar en este fructuoso mundo, lo consiguió de una manera espectacular lanzando al mercado lo que muchos definieron como un PC en forma de consola, lo que en parte era cierto y una ventaja al poder portar los juegos de PC a la consola gracias a las librerías *DirectX*[13] de Microsoft. Esta nueva consola se llamaría *Xbox*. Basada en un procesador *Intel Pentium III*, fue la primera en incorporar un disco duro para salvar partidas y extras, lo cual fue un gran atractivo para los jugadores. La filosofía de *Microsoft* de más vale lo bueno conocido que malo por conocer es lo que les dio muy buen resultado, siendo una consola muy exitosa que plantó cara a la todopoderosa PS2. La figura 47 muestra cómo era la máquina.

Figura 47. Xbox de Microsoft

En **2004** Nintendo lanzó *Nintendo DS (Dual Screen)* [36], una videoconsola portátil de séptima generación que como su propio nombre indica, su característica principal radica en la doble pantalla, las cuales se complementan y permiten dos puntos de vista de un mismo lugar, acceder a mapas y menús de forma rápida y fácil.

Otra característica interesante de esta consola es la pantalla táctil, pionera en este ámbito, que permite interactuar al jugador con la máquina como nunca se había hecho en una consola, acercándola más a una PDA que al resto de las consolas. La pantalla inferior de la Nintendo DS está superpuesta con una pantalla resistiva lo cual la hace un tanto imprecisa. La otra pantalla es un LCD estándar sin función táctil. El uso de dos pantallas produce mayor complejidad en los programas de software al especializar las funciones de cada pantalla. En Nintendo DS, algunos juegos vienen configurados para poder ser usados por zurdos o usando la máquina en posición vertical en vez de apaisada.

Por último, la posibilidad de conectarse a internet vía Wi-Fi, permite al jugador un amplio abanico de posibilidades.

Posteriormente aparecieron las versiones DSi y la DSi XL como evoluciones de ésta, pero no con tanto éxito como la original. En la siguiente figura se aprecian las características de esta maravilla.

Figura 48. Nintendo DS

No fue hasta 3 años después, en **2005**, cuando Sony lanzó su primera consola portátil, la *PlayStation Portable (PSP)* que con su gran pantalla panorámica, la posibilidad de poder jugar en modo multijugador a través de Wi-Fi y una gran capacidad en tan poco espacio, consiguió plantar cara a Nintendo, que hasta el momento tenía el monopolio en este terreno. La imagen de la figura 49 muestra el formato de esta consola portátil.

Figura 49. PlayStation Portable

Este mismo año, Microsoft, tras el éxito obtenido con Xbox y con la colaboración de **IBM**, con quien diseñaron la nueva arquitectura, lanzó la nueva *Xbox 360*, mucho más potente que su antecesora. Fue la primera consola de la actual generación, y la primera que sacó los famosos mandos sin cables, lo que le dio a *Microsoft* una gran cantidad de juegos que, al momento de salir sus competidoras, ya estaban empezando a sacar todo su potencial, siendo de las más vendidas del momento. Ofrece compatibilidad con juegos de su predecesora, aunque a modo de emulación, por lo

que muchos no funcionan correctamente. Fue una máquina que se vendió prácticamente a precio de coste, con un margen de beneficio muy ajustado en una estrategia comercial a largo plazo (técnica que más adelante copiaría Sony). En cambio Microsoft, fiel a su tradicional estilo de desarrollo rápido y descuidado, dejó algunos fallos en la máquina que provocan, de forma aleatoria, que la consola quede inservible. En la figura 50 se muestra la consola.

Figura 50. Xbox 360

En cuanto a consolas se refiere, hasta la actualidad, podemos destacar dos consolas que salieron paralelamente en el año **2006** y que a día de hoy, junto con la Xbox 360, se puede decir que son las videoconsolas que predominan en los hogares de la actualidad, estamos hablando de la famosa *Wii* de Nintendo y la súper esperada *PlayStation 3* de Sony.

Las promesas de una nueva forma de jugar a los videojuegos no hicieron más que despertar la curiosidad de todos. Finalmente se presentó un prototipo y se pudo ver dónde estaba la idea revolucionaria. en el mando. La consola prescinde de cables, necesitando una banda de infrarrojos instalada en el televisor. Además, son 2 mandos, uno para cada mano, pero con funciones diferentes. Básicamente la idea está que la forma de coger los mandos se adapta a cada juego, generando una experiencia totalmente nueva y más intuitiva. En cuanto a potencia, es la más modesta de su generación, habiéndole dado más importancia a su forma de juego. Eso automáticamente descarta a los grandes jugadores, pero en cambio atrae a los jugadores casuales, mercado en el que *Nintendo* pretendía entrar. La consola vendió muchas más unidades que sus competidores en el momento del lanzamiento y los meses siguientes y el famoso Wii Sports, del cual hablaremos más tarde, causó una auténtica revolución en los juegos de deportes.

Por su parte, Sony necesitaba contrarrestar el éxito de Xbox 360 con una nueva consola que supliera las carencias de PS2, de esta forma nació PlayStation 3 [37] con el inconveniente de que ahora tenía dos serios competidores en escena. A diferencia de su predecesora, la *PlayStation 3* ya no

reinaba a sus anchas. Se trata de una consola clásica, como lo es la *Xbox 360*, con unas capacidades técnicas altísimas, incluyendo el procesador que *IBM* creó especialmente para la consola, el *Cell*. Como de costumbre, esta versión de *PlayStation* incorpora el último avance en almacenamiento, en este caso no es otro que *Blue-ray*, volviendo a ser una máquina más económica que los primeros lectores del mercado. La máquina salió a la venta a un precio menor que su coste de fabricación, provocando pérdidas millonarias a *Sony*, que tuvo que subir el precio del resto de sus productos electrónicos para compensar.

Posteriormente salió una nueva versión más simplificada que eliminaba la compatibilidad con sus predecesoras, lo cual despertó fuertes críticas. Las siguientes figuras muestran ambas consolas.

Figura 51. Wii de Nintendo

Figura 52. PlayStation 3 de Sony

Aquí se ha dado fin al repaso por la historia de las videoconsolas desde los orígenes a la actualidad. Hasta el momento se pueden distinguir siete generaciones de videoconsolas, donde las que más impacto causaron se pueden ver en el siguiente cuadro.

Generación	Año	Consolas
Primera generación	1972/1979	Magnavox Odyssey 100, Magnavox Odyssey 200, Atari
Segunda generación	1977/1984	Atari 2600, Atari 5200, Arcadia 2001
Tercera generación	1982/1992	Atari 7800, NES, Master System
Cuarta generación	1988/1996	Mega Drive, Neo-Geo, Super Nintendo, Game Boy, Game Gear
Quinta generación	1993/2002	Saturn, PlayStation, Nintendo 64
Sexta generación	1998/2006	DreamCast, PlayStation 2, Xbox
Séptima generación	2005/Actualidad	Xbox 360, PlayStation 3, Wii, Nintendo DS, PSP

Tabla 1. Generaciones de consolas

Por último, se realizará un repaso por los mejores videojuegos de esta primera década del siglo XXI empezando por el mítico *Halo* [38] desarrollado por **Bungie Studios** en **2001**.

Se trata de un juego de acción en primera persona que ha recabado numerosos premios desde que salió al mercado, siendo considerado uno de los mejores juegos del género, lo cual queda refutado en las casi 10 millones de copias que se han vendido hasta el momento. Las características del juego se basan principalmente en que Halo lleva al jugador al universo de la ciencia ficción directamente desde una película de Hollywood con una línea argumentativa retorcida y detallada, personajes complejos y enemigos astutos; permite la lucha online con otros jugadores de cualquier parte del mundo en una gran variedad de juegos por equipo o a nivel individual; permite elegir entre una gran variedad de armas y misiones; por último, la compatibilidad con el sistema 5.1 de audio envuelve al jugador en un entorno absolutamente realista. En la siguiente figura se muestra una escena del juego.

Figura 53. Escena de batalla en Halo

Se debe destacar en este año el lanzamiento por parte de Konami del inicio de la saga *Pro Evolution Soccer (PES)*, un juego de simulación de fútbol que desde 2001 saca anualmente una nueva versión, y todas con el mismo éxito que en sus inicios. Los jugadores más fieles son los usuarios de PC's, ya que la jugabilidad facilitada con el teclado era algo que en las videoconsolas no destacaba, aunque actualmente las nuevas versiones salen en prácticamente todos los soportes posibles. La característica principal del juego radica en los gráficos y rendimiento, algo que su competidor, FIFA, era su punto débil.

En **2004** aparece una secuela del exitoso GTA, el *GTA San Andreas*. Explorar una gran ciudad no es fácil. San Andreas se convirtió en unos de los juegos buque insignia de PlayStation 2 donde se explota al máximo el recurso de hacer lo que al jugador le venga en gana, y la misión principal si se desea puede esperar. Otros aspectos a mencionar son el contenido adulto y violento del juego por lo cual tuvo bastantes críticas. La figura 54 muestra el contenido violento del juego.

Figura 54. Violencia en GTA San Andreas

También en 2004 vio la luz tal vez el mejor juego de la saga Metal Gear Solid de Konami en su tercera versión, *Snake Eater*. En ésta entrega, tanto el sistema de juego como la línea argumental cambian radicalmente, aun permaneciendo fieles a los títulos anteriores. Al desarrollarse la acción en 1964, las armas, vestimentas, pensamientos políticos e intrigas son típicos de la Guerra Fría. Para poder avanzar en el juego, el jugador debe adaptarse a las situaciones en las que el personaje se ve inmerso, debe obtener el armamento, los alimentos y las medicinas de su entorno, es necesario explorar los mapas para encontrar armas y recargas, cazar para poder alimentarse, y buscar plantas curativas para conseguir medicinas. La trama, el sonido, los detallados gráficos y el nuevo sistema de juego y de cura hicieron de esta nueva entrega de las aventuras de Snake un juego atractivo y atrayente para los aficionados a la saga. La figura 55 muestra la carátula del juego para PS2.

Figura 55. Carátula de Metal Gear Solid 3

Aun así, el juego más importante del año por su repercusión fue *World of Warcraft* [39], y fue el juego más vendido de PC desde 2005 hasta 2006. Fue aclamado universalmente por la crítica después de ser lanzado, siendo precedido de un período de grandes expectativas antes de su publicación.

Aunque el juego sigue un modelo similar a otros en el género y usa conceptos de los juegos de rol las nuevas propuestas para reducir las pausas entre encuentros fueron bien recibidas. La personalización física de los personajes era lo menos destacado del juego, aunque se alabó el nivel de detalle de los modelos. La apariencia del juego fue ensalzada por los críticos y funcionaba bien en casi cualquier tipo de sistema.

Al año siguiente, en **2005**, llegaron al mercado dos títulos para PS2 muy atrayentes y con mucho éxito, aunque de géneros muy distintos como son *God of War* [40] de **SCEA** y *Guitar Hero* de **Harmonix Music System**.

El primero es un juego de acción que narra las aventuras de Kratos, un general espartano al servicio de los Dioses de la Mitología griega. El desarrollo del juego gira en torno a la idea de la hybris de Kratos al rebelarse contra los dioses y su propio destino y la némesis a la cual se ve sometido, estando así en consonancia con la literatura griega clásica. Sin embargo, en este caso la hybris de Kratos no sólo no es castigada sino que le supone una victoria y destino mayor del que él mismo había concebido. Kratos es el destructor de Dioses.

Los espectaculares gráficos tanto de las escenas como de los personajes dotaron al juego de un éxito que superaba a los de la competencia como se puede ver en la figura 56.

Figura 56. Escena de God of War.

Guitar Hero [41] por su parte, es un juego de música inspirado en la mecánica de juego Dance Revolution que sorprendió a la industria del videojuego. Tal fue el éxito que dio lugar a nuevos fenómenos como *Rock Band* o el *Band Hero*.

Guitar Hero incluye un controlador en forma de guitarra eléctrica similar a una guitarra **Gibson SG** utilizada en vez de un mando convencional, que tiene botones que simulan los trastes y un controlador que simula una palanca vibradora. Este nuevo modelo de juego tuvo y tiene gran aceptación entre los aficionados a la música gracias al realismo de simular a las estrellas del rock pese a tratarse de un juego. Posteriormente salió una versión con batería y micrófono, el *Guitar Hero World Tour*, en el cual se podía emular a un grupo de música completo. Cuando se juega en modo experto, el parecido a tocar una batería de verdad es prácticamente el mismo, de ahí el éxito rotundo de este título. En la figura 57 se muestran la guitarra y la batería que vienen en el juego.

Figura 57. Accesorios de Guitar Hero

En **2006** Nintendo revolucionó el mercado como se ha visto antes con su videoconsola Wii, y de la mano el videojuego *Wii Sports*. El juego consta de cinco deportes. tenis, béisbol, boxeo, golf y bolos. En cada uno de ellos el mando de la Wii simula ser una raqueta, un bate de béisbol, un guante de boxeo, un palo de golf y una bola de bolos, respectivamente.

Este modelo de juego no destaca por la calidad de los gráficos ni el audio, ni siquiera por los personajes, sin embargo, si lo hace en cuanto a la revolucionaria nueva forma de jugar, absolutamente dinámica en la que es el propio jugador quien realiza los movimientos del personaje.

En **2007** el juego que lideró las ventas fue de nuevo un éxito de Nintendo para Wii, y de nuevo una secuela del mítico Super Mario, con *Super Mario Galaxy*, tal vez el mejor juego de Mario de la década para muchos. En esta entrega se introduce en este juego la revolución del género de plataformas el cual no respeta la ley de gravedad y nos muestra un mundo totalmente en 3D lleno de colorido y diversión al máximo. A ello hay que agregarle las múltiples acciones que se pueden realizar con el mando con sensor de movimiento de Wii. La figura 58 muestra una imagen del juego.

Figura 58. Secuencia de Super Mario Galaxy

Durante los últimos dos años, las empresas, más que abordar nuevos proyectos están siendo conservadoras y lanzan secuelas de sus mayores éxitos, asegurándose la aceptación de los seguidores de las sagas en la época actual de crisis mundial. Tal es así que en **2008** pudimos ver el lanzamiento de *Fallout 3*, cuya principal característica es que permite personalizar el carácter del jugador al máximo, además de contar con un amplio arsenal que también se puede personalizar. A ello hay agregarle la infinidad de personajes y escenarios que ambientan a la perfección un mundo acabado por la guerra nuclear. Cabe mencionar que Fallout 3 tiene una banda sonora sorprendente con música de los años 30, pero lo más valorado por los jugadores son sus más de 40 horas de juego.

En **2009** llegaron *God of War 3* y *Halo 3* y en **2010** se prevé el lanzamiento de *Max Payne 3* y la decimotercera entrega de *Final Fantasy* como aspectos más destacados.

Como se ha visto durante el repaso a la historia de los videojuegos, se puede observar que en un periodo relativamente corto de tiempo, esta industria ha ido evolucionando a pasos agigantados, llegando a convertirse en uno de los motores de la economía actual en un país tan importante como Estados Unidos. De esta manera, **Doug Lowenstein**, presidente de la Asociación del Software para en Entretenimiento concluyó en su estudio titulado "*Videojuegos. Un asunto serio para la economía estadounidense*" [42] que las ventas de programas informáticos de juegos electrónicos alcanzarían en Estados Unidos los 15.000 millones de dólares y la industria contaría con más de 250.000 empleos altamente cualificados para 2010.

2.2.3. Los videojuegos en la actualidad

Actualmente, el mundo de los videojuegos es un mercado ya consolidado, y la tecnología avanza muy rápido, entre otras cosas debido a la gran competencia que existe ya que todas las empresas buscan diariamente algo con lo que sorprender a sus usuarios, además de buscar nuevas maneras de atraer a determinados tipos de clientes que nunca han tenido curiosidad por los videojuegos.

El desarrollo de los videojuegos en los últimos años se ha caracterizado por la diversificación de la oferta en lo relativo al contenido y desarrollo del juego, e incluso en su grado de relación con la realidad. En la actualidad, existen diferencias notables entre unos juegos y otros en cuanto a las habilidades y recursos psicológicos necesarios para su utilización, estrategias de resolución de problemas, establecimiento de relaciones causales, toma de decisiones, etc. La séptima generación de videojuegos incorpora representaciones gráficas complejas de alta resolución, el uso de personajes reconocibles y un apartado sonoro atronador. Además, se otorga al jugador un notable grado de libertad a la hora de seleccionar o manipular los distintos elementos que intervienen en el juego.

Las consolas actuales están muy consolidadas en el mercado, y los fabricantes buscan dar utilidades a sus potentes máquinas, como es por ejemplo el caso de **Kinect** para Xbox360.

Kinect [43] es un periférico destinado a jugar sin mandos cuya característica principal es la posibilidad de hablar con él, aunque esta opción de momento sólo es aplicable al mundo anglosajón, y aún está en pruebas. Kinect ofrece la posibilidad de jugar con todo el cuerpo. Se pueden utilizar los brazos, las piernas, los pies y las manos. Es un modo de juego muy intuitivo ya que los movimientos del jugador son exactamente los que el actor del juego realiza, por lo que dota al jugador de una capacidad hasta ahora nunca conocida.

Las principales habilidades de Kinect son las siguientes:

- ✓ Uso de todo el cuerpo del jugador (cabeza, tronco y extremidades)
- ✓ Detecta las propias herramientas del jugador, como por ejemplo un monopatín
- ✓ Reconocimiento facial para reconocer a cada jugador
- ✓ Conexión a internet
- ✓ Reconocimiento de voz

En la figura 59 se pueden ver los componentes de Kinect.

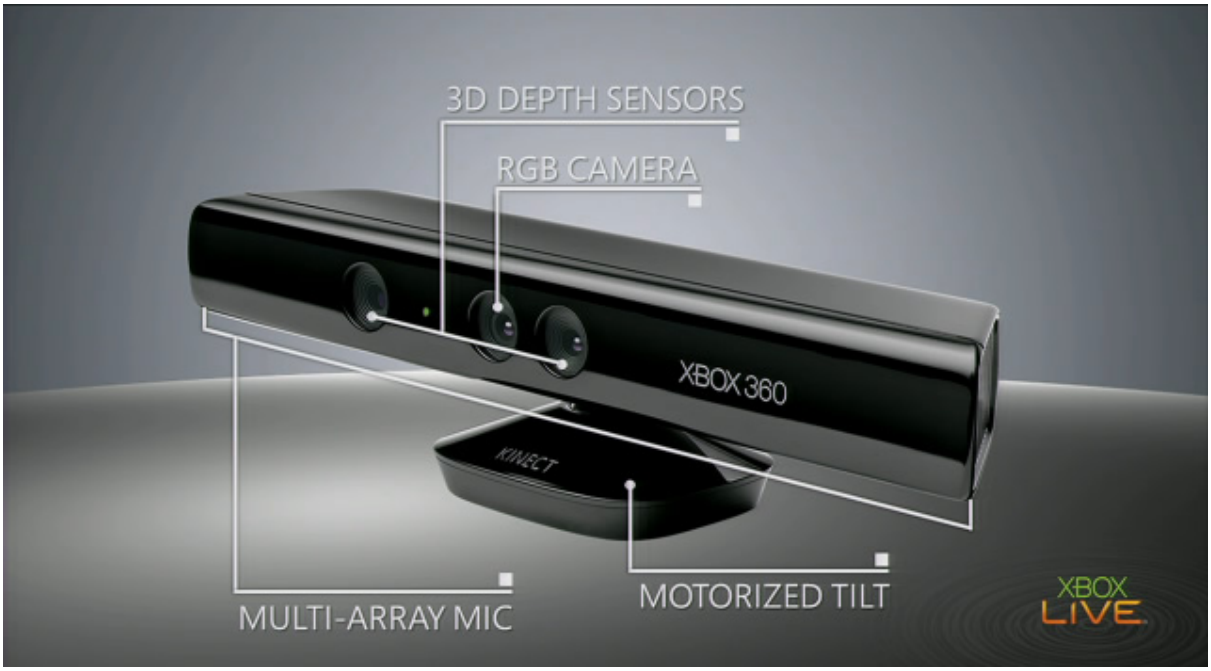

Figura 59. Componentes de Kinect

Sony por su parte se debate en mil batallas contra los hackers, y actualmente trata de conseguir una PS3 y una Wii más seguras y más difíciles de piratear; mientras tanto su potencial radica en mantener a sus fieles y un buen ejemplo es potenciando uno de sus mejores y con mayor número de seguidores, como es el Final Fantasy XIII. Según su creador, Motomu Toriyama, superar la historia principal puede costarles a los jugadores expertos más de 50 horas, todo un reto desde luego.

Uno de los problemas más importantes con los que luchan las empresas fabricantes de consolas, como hemos dicho antes, es la piratería en modo de chip, a lo que se le llama **modchip** o chip modificado [44]. Este chip es un circuito integrado que se instala o accede vía usb o por otro puerto en forma de pequeño dispositivo electrónico. El modchip es utilizado para modificar o desactivar las restricciones y limitaciones impuestas por las empresas fabricantes de consolas de videojuegos más populares (Nintendo, Playstation, Xbox, etc). El modchip introduce diversas

modificaciones en función del sistema al cual se instala, incluyendo la elusión de la región de codificación, gestión de derechos digitales, y los controles de protección contra copia con el fin de ejecutar software destinados a otros mercados, los medios de comunicación, copias de respaldo de los videojuegos, o software de terceros no autorizado.

Los modchip se utilizan principalmente en sistemas de videojuegos basados en CD/DVD, debido a la disponibilidad. Además de los juegos de consolas, modchips también están disponibles para algunos reproductores de DVD para eludir la ejecución de código de región y prohibiciones de operación del usuario.

En este sentido destaca la empresa **PS Jailbreak** [45] y su modchip con el que se consigue copiar un juego desde su disco original al disco duro interno de la consola o a uno extraíble y posteriormente ejecutarlo sin volver a necesitar el disco original.

Por último, cabe destacar una nueva forma de jugar que aún no tiene muchos adeptos, pero a buen seguro que conseguirá atrapar a muchos por su sencillez, esta nueva forma de jugar se le conoce como **Streaming**.

La plataforma **OnLive** [46] supone una revolución en el mercado de videojuegos que permite jugarlos a través de la web sin necesidad de un ordenador potente, bastando con uno de clase media para ejecutar el programa base de obligatoria descarga. Este tipo de tecnología es la que se espera que se adopte en el futuro, de hecho ya hay compañías interesadas.

OnLive utiliza una tecnología de compresión y servidores de alta calidad que procesan el juego en los servidores y lo sirven en el del usuario sin esperas. El servicio sí que requiere una buena conexión de banda ancha, al menos con 5 megabytes de descarga, recomendándose además de evitar jugar a través de Firewalls corporativos que lo enlentecerían. De este modo sería posible conectarse a la cuenta de OnLive desde cualquier ordenador, siempre que permita la instalación del programa base y eso sí, pagando una tarifa mensual.

En todo caso, está por ver si la distribución convencerá a los jugadores, ya que no ofrece producto físico y sí unos precios similares al actual modelo de negocio de las consolas.

2.2.4. Futuro de los videojuegos

Hoy en día parece que ya está todo inventado y que ya nada puede sorprender, sin embargo, la industria del videojuego galopa a toda velocidad y no sólo nos sorprende con cosas como las vistas en el apartado anterior, sino que piensan y trabajan en el futuro para seguir sorprendiéndonos.

Cualquier pronóstico sobre el futuro es arriesgado, es muy probable que la realidad acabe desbordando nuestras previsiones, que las líneas de trabajo que hoy parecen prometedoras acaben en humo y que se pasen por alto temas importantes.

Se ha visto en el repaso a la historia de los videojuegos la evolución desde un juego tan simple como el Pong, a lo último en tecnología como son los juegos a través de Kinect, sin embargo, aunque no se puede deparar lo que se verá a medio y largo plazo, si es posible hablar sobre algo en lo que históricamente se está trabajando y que hoy en día tiene un futuro prometedor como es la **realidad virtual**.

Desde el punto de vista de la tecnología, la realidad virtual [47] puede definirse como la suma de los sistemas de hardware y de software que aspiran a construir una ilusión sensorial de estar presente en otro ambiente, en otra realidad. El proceso de creación de esta sensación de presencia comienza con el emparejamiento de los órganos sensoriales del usuario a los elementos de salida de la CPU. Los ojos, las orejas, las manos y los sensores propioceptivos reciben estímulos electromecánicos que intentan simular la acción de un mundo ficticio sobre los sentidos. Todo el cuerpo del jugador se convierte en un puerto de entrada, al tiempo que todos los movimientos conscientes e inconscientes del cuerpo se convierten en datos de entrada para la máquina.

La industria lleva varios años trabajando en el desarrollo de sistemas de representación visual acoplados a la cabeza que proporcionen estímulos visuales, sistemas de audio espacial que aporten estímulos auditivos, sistemas de retroalimentación táctil que simulen los estímulos propioceptivos, tales como la sensación de fuerza. Otros elementos deben recoger los movimientos del sujeto, su voz y las respuestas de su sistema nervioso autónomo para transmitirlos a la CPU.

En el año **1995**, Nintendo lanzó un novedoso sistema de juego con el llamado *Virtual Boy* [48] de 32 bits, como se ve en la figura 60. Con este juego, Nintendo trató de abrirse mercado en plena madurez de la Super Nintendo, pero con la inminente aparición de las consolas de 32 bits como PlayStation y Saturn, la compañía japonesa aunque lanzó al mercado japonés esta consola, nunca llegaría a Europa a causa del escaso éxito. Fue una novedosa máquina semi portátil parecida a los cascos de realidad virtual de los salones recreativos que, gracias a un trípode, se situaba encima de

una mesa y el jugador tenía que aproximarse a los visores para apreciar un efecto tridimensional en los juegos.

Figura 60. Virtual Boy de Nintendo

Todas las imágenes tenían un fondo negro, estando los objetos en gamas de rojo. Esta tonalidad podía ser cambiada mediante el uso de un curioso periférico, unas gafas, a gama verde o azul. Nintendo declaró que no incluyó otros dos LED (uno azul y otro verde para componer las imágenes en tonalidades reales) porque de esta forma la imagen era tres veces más nítidas y el proyecto tres veces más barato.

Este sistema fue recibido por la prensa con gran escepticismo, pues traía bajo el brazo varias polémicas. En primer lugar, su nombre, no se ofrecía la inmersión en un mundo independiente por el que moverse y con el que interactuar al cien por cien, que es lo que significa las palabras realidad virtual. Esta explicación fue motivo de grandes aclaraciones en toda la prensa mundial.

Sólo podía jugar una persona, que además estaba aislado del resto por el visor. Ni siquiera nadie puede ver el desarrollo del jugador desde el exterior. Para paliar esta deficiencia, Nintendo desarrolló el *Playlink*, para conectar dos sistemas, pero no fue una medida suficiente, pues el concepto de juego estaba en la máquina en sí.

En cuanto a las ventas, era muy difícil vender algo de lo que ni siquiera se podían captar imágenes (todas las que la prensa mostró eran simulaciones). Era necesaria la toma de contacto en las tiendas, por lo que el público potencial se reducía bastante.

A pesar de sus inconvenientes, la Virtual Boy fue un nuevo concepto en el mundo de los videojuegos. El efecto 3D se consiguió emular perfectamente con unos gráficos algo sencillos pero con una gran fluidez de movimientos. A causa de los pocos juegos que llegaron a comercializarse, el potencial de la máquina apenas fue aprovechado ya que los juegos existentes resultaron ser algo simples y lineales.

A día de hoy, la Virtual Boy resulta una pieza rara de. El intento frustrado de Nintendo por introducir la realidad virtual en el mundo de las consolas fue abandonado para dedicarse plenamente al desarrollo de la siguiente generación de consolas (Nintendo 64).

La realidad virtual puede ser de dos tipos. ***inmersiva*** y *no* ***inmersiva*** [49]. Los métodos inmersivos de realidad virtual con frecuencia se ligan a un ambiente tridimensional creado por un ordenador, el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano. La realidad virtual no inmersiva también utiliza el ordenador y se vale de medios como el que actualmente nos ofrece Internet, en el cual podemos interactuar en tiempo real con diferentes personas en espacios y ambientes que en realidad no existen sin la necesidad de dispositivos adicionales al ordenador. De este último tipo hoy en día cabe destacar lo último en realidad virtual con **Second Life** [50]. En Second Life no hacen falta cascos ni guantes, pero sí una conexión de banda ancha. El usuario tampoco está aislado. Todo lo contrario.

Su traducción sería "*Segunda Vida*", es un mundo virtual 3D de interacción social creado por **Linden Lab** y fundado por **Philip Rosedale**. Es un mundo que está distribuido en una amplia red de servidores y al que se puede acceder a través de Internet. Este programa proporciona a sus usuarios o "*residentes*" herramientas para modificar el mundo y participar en su economía virtual, que opera como un mercado real. Existe asimismo una versión para adolescentes, **Teen Second Life**. Esta versión fue desarrollada a principios de 2005. La media de edad de los residentes de este mundo virtual es de 32 años.

Second Life desciende directamente de la nueva generación de videojuegos conectados. Según esta definición, Second Life es un MMOG[14], o juego online multijugador masivo. Pero también es mucho más que un juego. En Second Life no hay misión, no hay orcos que degollar ni zombies que ametrallar. A cambio, hay dinero, negocios, juegos, cultura y sexo.

Cualquiera puede hacerse ciudadano de Second Life gratis. Basta con registrarse en su página web, descargar e instalar el programa. Con esto, y una buena tarjeta gráfica, se puede entrar en un mundo donde casi todo es posible, desde convertirse en un yeti peludo de tres metros de altura, hasta construir una réplica del Partenón y utilizarlo para dar una fiesta con música en vivo.

Como curiosidad, en 2008 en Londres, una pareja se divorció después de que la mujer se enterara que su marido había incumplido su voto de fidelidad conyugal, al tener una relación con una supuesta "*prostituta*" virtual.

Lo más gracioso es que esta pareja británica se conoció a través de Internet en mayo de 2003, se casaron en julio de 2005 en la vida real y también celebraron su boda en Second Life. De hecho, la esposa sospechaba desde el 2007 que su marido le era infiel e incluso llegó a contratar a un investigador virtual para seguirlo.

En la figura 61 se puede observar una imagen de la vida virtual de Second Life.

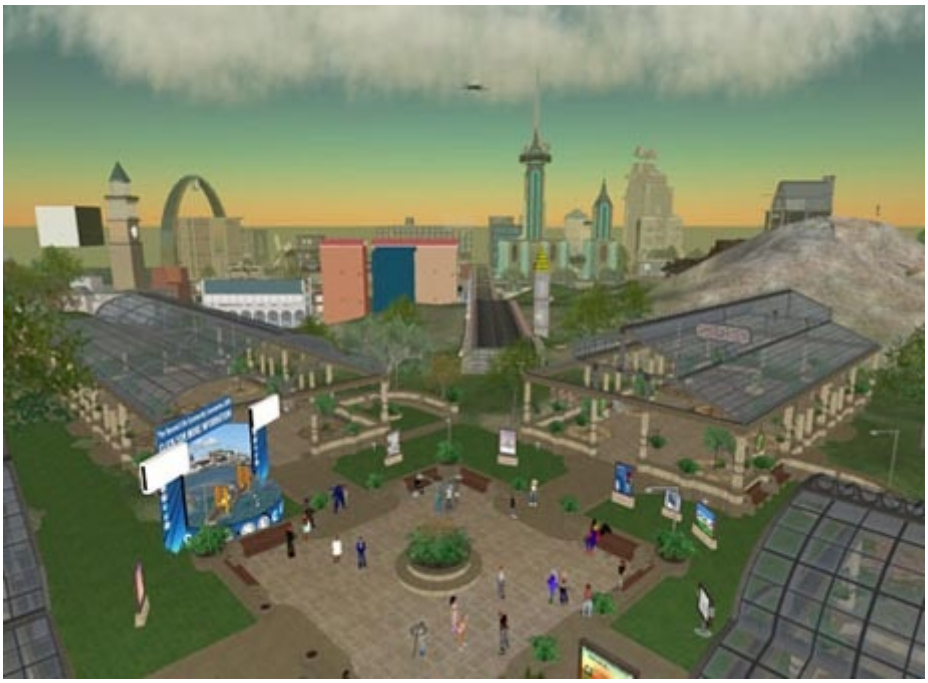

Figura 61. Imagen de Second Life

La evolución de los sistemas de realidad virtual seguirá un camino lento, pero no cabe duda de que aportará al mundo de los videojuegos la sensación de realidad a la que han ido aspirando las sucesivas innovaciones técnicas que hemos ido contemplando a lo largo de las últimas décadas**,** una sensación de realidad que está en la base misma de la actividad del videojuego.

2.3. Ciclo de vida de los videojuegos

El ciclo de vida da la pauta a lo que hay que obtener a lo largo del desarrollo del juego. Dada la complejidad de un videojuego, el proyecto debe de manejarse a través de un proceso, involucrando así una serie de pasos organizados que guiarán cada actividad hasta el producto final.

Desde que surge la idea inicial hasta que el producto se comercializa se atraviesan varias fases en las cuales intervienen distintos actores cada cual con capital importancia en su terreno.

El proceso de ciclo de vida es similar a la creación de software en general, aunque difiere en la gran cantidad de aportes creativos (música, historia, diseño de personajes, niveles, etc.) necesarios. El desarrollo también varía en función de la plataforma objetivo (PC, móviles, consolas), el género (estrategia en tiempo real, aventura gráfica, etc.) y la dimensión de visualización (2d, 2.5d y 3d).

Básicamente las fases por las que pasa un video juego son las siguientes [51]:

- Concepción de la idea del videojuego
- Diseño
- Planificación
- Producción
- Pruebas
- Mantenimiento

2.3.1. Concepción de la idea

En esta etapa es necesario definir los aspectos generales y fundamentales que conformarán el videojuego. Es la primera etapa en la que se forja la vida del videojuego, por lo que es de vital importancia hacer unos buenos cimientos para el producto. Cuando surge la idea por tanto, hay que tener en cuenta los siguientes factores:

- **Género**. Hay que especificar el género para que el jugador que compre el producto sepa a qué va a jugar. Los videojuegos se pueden clasificar como un género u otro dependiendo de su argumento, representación gráfica, el tipo de interacción entre el jugador y la máquina, la ambientación y su sistema de juego, siendo este último el criterio más habitual a tener en cuenta.

- **Jugabilidad**. Más usado el término en inglés *Game Play*, la Jugabilidad representa el grado en el que jugadores alcanzan metas específicas del videojuego con efectividad, eficiencia, flexibilidad seguridad y especialmente satisfacción en un contexto jugable de uso. Las características más importantes para que un videojuego tenga éxito en este sentido son las siguientes:
    - **Satisfacción.** Debe conseguir el agrado del jugador ante el videojuego completo o en algunos aspectos concretos de éste, como mecánicas, gráficos, sistema interactivo, historia, etc.
    - **Aprendizaje.** Debe ser fácil de comprender y dominar el sistema y la mecánica del videojuego, es decir, los conceptos definidos del juego. objetivos, reglas y formas de interaccionar con el videojuego.
    - **Efectividad.** Un videojuego efectivo es aquel que es capaz de captar la atención del jugador desde el primer instante, convenciéndole de que siga jugando al juego. A la vez, el juego con alto grado de efectividad debe entretener, incluso llevadas muchas horas de juego.
    - **Inmersión.** La inmersión es la característica del juego relacionada con provocar que el jugador se vea envuelto en el mundo virtual, volviéndose parte de éste e interactuando con él. En ese momento el jugador se hace cómplice de la mentira del mundo virtual, haciéndola verdad y produciéndose una inversión de creencia, la cual provoca que el jugador, aunque sepa que a lo que juega es falso, lo tome como algo real y se implique en ello con todas sus habilidades para superar el reto propuesto, estando concentrado o envuelto en la tarea propuesta por el videojuego.
    - **Motivación**. Para conseguir una buena motivación, el juego debe disponer de un conjunto de mecanismos que generen una perseverancia en la acción por parte del jugador para superar los retos del juego, es decir, se introducen factores que aseguren el mantenimiento de un comportamiento en la apreciación del proceso de juego.
    - **Emoción**. Los juegos generan distintos estímulos durante la dinámica del juego para desencadenar reacciones involuntarias automáticas y distintos sentimientos y emociones por parte del jugador para modificar su actitud y comportamiento cuando juega. alegría, presión, frustración, miedo, intriga, curiosidad… para construir un universo virtual capaz de conmover, emocionar, hacer sonreír o llorar al jugador si es necesario.

- **Socialización**. La socialización de un juego permite a un jugador tener una experiencia de juego totalmente distinta cuando juega un juego solo o en compañía de otros jugadores y fomenta nuevas relaciones sociales interactuando con ellos, ya sea de manera competitiva, colaborativa o cooperativa. Por otro lado, la socialización también está presente en cómo se proyectan las relaciones sociales que tenemos con el grupo en los personajes del videojuego, como puede ser el contexto en el que se realiza el juego, para obtener información, pedir ayuda, negociar en la compra/petición de objetos, la implicación de otros personajes o si les beneficia que el jugador cumpla los objetivos del juego.

- **Guión gráfico**. El guión gráfico o más conocido como "*story board"* es el conjunto de ilustraciones que se muestran de manera secuencial para entender el videojuego antes de comenzar a programarlo. En el guión gráfico se muestran ideas acerca de cómo debe lucir el juego en cuanto a personajes, ambientación, música, etc. Suele ser muy usado por los guionistas publicitarios. En la figura 62 se muestra un guión gráfico de uno de los anuncios más conocidos de Coca-Cola.

Figura 62. Guión gráfico del anuncio "La fábrica de la felicidad" de Coca-Cola

2.3.2. Diseño

El diseño es la fase en la que se van a detallar todos los elementos y componentes que van a dar forma al juego. Todo debe ser lo más claro y sencillo posible para que posteriormente el equipo de desarrollo pueda realizar su trabajo lo más fielmente posible a la idea original. En el diseño de un videojuego se han de especificar muchas cosas, pero entre las más importantes y por orden cronológico podemos destacar las siguientes:

- **Historia**. La historia del juego cuenta a grandes rasgos el argumento y la manera en la que los personajes se van a desenvolver a media que el juego avance.
- **Arte conceptual**. En esta parte entran en escena los diseñadores gráficos para establecer el aspecto general del juego. Los diseñadores se encargan de crear los personajes, los escenarios, los objetos, etc. Posteriormente entregarán las

propuestas en formato digital como boceto de cómo sería el juego y será director de arte el que decida lo que se ajusta más fielmente a la idea original.

- **Sonido**. La elección del sonido, música, banda sonora y efectos de sonido son elementos clave en el éxito del videojuego hasta el punto que juegos del tipo *Call of Duty* o *God of War*, deben gran parte de su éxito a esta faceta. Es tal la importancia que desde hace unos cuantos años, la **Game Audio Network Guild** (G.A.N.G.), una organización sin ánimo de lucro que se dedica a la promoción del audio en los videojuegos otorga premios a los mejores juegos del año en cuanto a sonido se refiere.
- **Mecánica de juego**. La mecánica de juego consiste en definir las especificaciones y las reglas del juego. Se encarga de definir cómo los personajes y objetos interactúan dentro del juego en todas las circunstancias posibles. Es una fase muy importante ya que no pueden quedar cabos sueltos cuando se programe el juego.
- **Diseño de programación**. La fase de diseño de programación está ligada a la elección de los elementos que se necesitan para desarrollar el software y es la más técnica del diseño. En ella se describe en qué soporte se programará el juego (PC, consola, móvil, etc), el lenguaje de programación que se usará y la metodología, el modelado gráfico y secuencial en que los personajes interactúan, etc. Finalmente se realiza un **documento de diseño técnico** en el que quedan reflejadas todos estos aspectos.

2.3.3. Planificación

La planificación es una de las fases más importantes del proyecto, ya que la viabilidad en los plazos de entrega del producto depende de si se ha hecho una buena planificación o no. En esta fase se identifican todas las tareas, se estima el tiempo que conlleva realizarlas y se reparten al equipo de desarrolladores.

Además de estimar los plazos, se establecen reuniones de seguimiento a los largo del proyecto para verificar si la planificación va por los cauces establecidos o es necesario re planificar, estudiando la razón de esta re planificación, ya que pueden ser causas inesperadas o por el contrario es que no se ha planificado bien desde el principio. Los jefes de proyecto realizan este trabajo y se ayudan de herramientas gráficas como los diagramas de **GANTT**[15] o **PERT**[16]. La figura 63 refleja un diagrama de GANTT con las tareas a realizar, los recursos que las llevarán a cabo y los tiempos estimados para la realización de éstas.

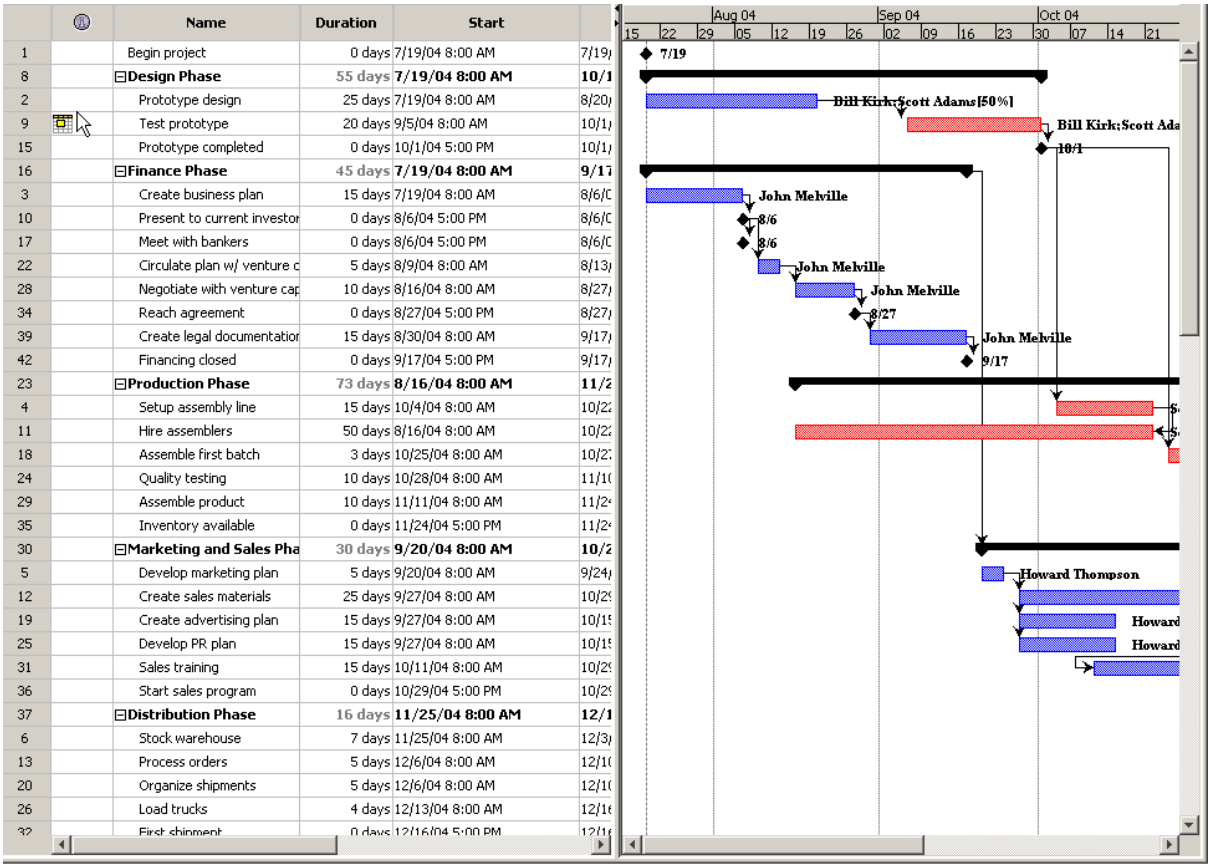

Figura 63. Diagrama de GANTT

2.3.4. Producción

Es la fase técnica del proyecto en la que se crean los personajes, se crean la música, voces y sonido; en resumen, es donde se realiza la programación del software del juego. La fase de Producción consta de varios aspectos en la que los integrantes del equipo de desarrollo deben interactuar conjuntamente hasta la finalización del producto. Estos aspectos son los siguientes:

- **Programación**. Los programadores tienen la responsabilidad de dar vida al proyecto. Para ello usan lenguajes de programación de alto nivel especializados en la creación de videojuegos. Dependiendo de los requerimientos del proyecto se usará el más adecuado. El lenguaje más utilizado para el desarrollo de videojuegos para consolas y PC's es **C++** mientras que para dispositivos móviles es más común usar **Java.** Los desarrolladores de juegos para Xbox usan el entorno de Visual Studio para programar en **C#** y se apoyan en el conjunto de herramientas (*Framework*) más popular de Microsoft, **XNA**.

- **Ilustración**. Los artistas gráficos se encargar de crear los personajes y los gráficos del juego. Además deben elaborar las animaciones y dotar al juego del mayor realismo gráfico posible. Una de las aplicaciones de interfaz de programación (API) más usadas por los programadores gráficos es **DirectX** ya que proporciona una colección de API's creadas para facilitar la programación multimedia en la plataforma Microsoft Windows. Entre las API's de DirectX más usadas por los programadores destacan **Direct3D** (utilizado para el procesamiento y la programación de gráficos en tres dimensiones), **Direct Graphics** (para dibujar imágenes en dos dimensiones (planas), y para representar imágenes en tres dimensiones) y **DirectSound** (para la reproducción y grabación de sonidos).
- **Interfaz**. Es la manera en la que los elementos de la Interfaz Gráfica de Usuario (GUI) serán vistas, a través de los cuales, el usuario interactuará con el juego. El uso principal del GUI consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina.
- **Modelado y animación 3D**. Los creadores de las animaciones y los modelados en 3D se ayudan de herramientas profesionales para su desarrollo tan populares como **XSI/Softimage**, **Blender** o **Maya** entre otras.
- **Diseño de Sonido**. Los ingenieros de sonido se encargan de crear todo lo referente a los diálogos entre los personajes, efectos de sonido y crean la banda sonora del juego. Un buen ingeniero de sonido es capaz de no sólo crear sonidos, sino también reflejar emociones, personificar un elemento de la historia del juego, dar significado a una escena o generar todo tipo de sensaciones en el jugador.

2.3.5. Pruebas

Una vez terminada la fase de Producción y el juego está terminado, llega la fase de Pruebas. Parece a simple vista una fase sencilla, sin embargo requiere de personal altamente cualificado para realizar la batería de pruebas y buscar fallos tanto técnicos como funcionales del juego. Las pruebas suelen llevarse a cabo en 2 fases, una primera fase (**alfa**) en la que un grupo de personas que han estado involucradas en el desarrollo del juego (artistas, programadores, etc.) realizan las primeras pruebas para detectar y corregir los fallos o defectos más visibles y mejorar características no contempladas en la fase de diseño; posteriormente en la segunda fase (**beta**), las pruebas las lleva a cabo un equipo externo de jugadores/probadores profesionales para conseguir un juego robusto y sin errores. Al finalizar estas pruebas el videojuego debe salir al mercado con la menor cantidad de defectos menores y ninguno medio o alto.

2.3.6. Mantenimiento

La fase de mantenimiento del software del videojuego es la última fase en el ciclo de vida de un videojuego hasta la retirada del producto del mercado. Durante esta fase, los ingenieros realizan un proceso de mejora y optimización del videojuego una vez ya ha sido comercializado; por tanto se trata de una revisión del producto constante para prevenir y corregir pequeños defectos.

Los videojuegos, como se ha dicho anteriormente no deja de ser un producto de software más, por lo que se ajusta a la misma metodología que un producto de software normal; por tanto, la fase de mantenimiento también tiene sus reglas según el tipo de mantenimiento que se realice, que pueden ser de cuatro tipos:

- **Perfectivo**. El mantenimiento perfectivo consiste en mejorar la calidad del software, tanto en la restructuración del código, como en la definición del sistema o la optimización del rendimiento y eficiencia.
- **Evolutivo**. A medida que avanza el tiempo del videojuego en el mercado, pueden surgir nuevas ideas o por el contrario funcionalidades que quedan obsoletas, por lo que en relación a lo que el mercado vaya demandando, en este caso los jugadores, se realizan las modificaciones, incorporaciones o eliminaciones necesarias para mantener el juego en un nivel competitivo.
- **Adaptativo**. En este caso son modificaciones que afectan al hardware como cambios en la configuración del hardware, como puede ser la incorporación del juego a una nueva consola por ejemplo.
- **Correctivo**. El mantenimiento correctivo trata de corregir pequeños defectos de software que hayan sido detectados después de la salida del videojuego al mercado.

Todos estos tipos de mantenimiento no son excluyentes, sino que se solapan hasta el punto de que cuando se detecta alguna anomalía, los desarrolladores lo solucionan y sale una nueva versión del juego. Generalmente este tipo de correcciones suelen "*taparse*" añadiendo nuevas funcionalidades al juego para hacerlo atractivo al jugador y compre la nueva versión con el defecto subsanado y a su vez dotándole de nuevos personajes, dificultades, etc, de esta manera se realizan varios tipos de mantenimiento a la vez.

2.4. XNA Game Studio

XNA es un conjunto de herramientas con un entorno de ejecución administrado proporcionado por Microsoft que facilita el desarrollo de juegos de ordenador. Está basada en DirectX y manejada por .NET[17]. Se trata de un API de programación que simplifica y hace más intuitivo el uso de las librerías nativas DirectX y, en consecuencia, simplifica de manera notable la programación de videojuegos.

2.4.1. Qué es XNA y por qué usarlo

Básicamente, XNA [52] se compone de dos librerías de ensamblados denominadas *dll's*[18] para .Net que manejan por debajo las librerías nativas del sistema de DirectX. Al situarse como una librería sobre el motor de .Net, se beneficia de todas las mejoras que .Net proporciona incluyendo el manejo de la memoria por el recolector de basuras, la posibilidad de trabajar con cualquier lenguaje .Net y la *multiplataforma*[19].

Previamente a XNA existían las librerías de ensamblados *Managed DirectX* que, de manera similar a XNA, permitían el acceso desde lenguajes .Net a las librerías nativas de DirectX. En este caso, sin embargo, el acceso era mucho menos intuitivo y se componía, básicamente, de un calco de las funciones de DirectX para .Net. Las librerías Managed DirectX están en desuso y no se soportarán más en el futuro.

El modelo que Microsoft ha propuesto se trata de un modelo de programación orientado justamente a la programación exclusiva de videojuegos, por lo que dota al producto de todas las capacidades posibles para facilitar el trabajo al desarrollador. Para ello se definen una buena cantidad de clases básicas de tipo matemático, necesarias para la programación gráfica, pero también una nueva API y metodología de programación sobre estas clases que simplifica el proceso de desarrollo.

El desarrollo de videojuegos con XNA está orientado a componentes. Los componentes son, básicamente, pequeños trozos de código que cumplen una funcionalidad concreta. Existen componentes gráficos como podría ser una bola rebotando por la pantalla, una nave espacial o el mismo escenario del videojuego. No obstante no todos los componentes tienen por qué ser gráficos. Podemos tener un manejador de eventos o estados del videojuego, un generador de lógica u otros tipos de componentes que no necesiten ser *renderizados*[20] para usarse y cumplir su función. Todos estos elementos heredan de la clase GameComponent (para los elementos no gráficos) o de la clase DrawableGameComponente (para los elementos gráficos).

Una clase central (Game) se encarga de manejar todos los componentes del juego (*GameComponents***)** que han sido registrados. Así, un videojuego (Guame) es igual a la suma de las funcionalidades de todos los GameComponents que lo componen.

En la actualidad, XNA utiliza un *IDE*[21] de programación basado en el IDE de C# 2005 Express Edition, por lo tanto la mayoría de los programas basados en XNA están escritos en C#. Sin embargo, al tratarse de librerías de ensamblados totalmente compatibles con .Net, pueden utilizarse con cualquier otro lenguaje .Net del mismo modo que se utilizaría cualquier otra librería de ensamblados.

Las tecnologías base de XNA, al ser compatibles totalmente con la estructura de programación de .Net, tienen la propiedad de ser también multiplataforma. Actualmente, ese es el terreno en el que se está trabajando con más esfuerzo ya que se pretende una compatibilidad 100% entre los Frameworks para Windows y para XBox 360. De momento, todavía existen algunas incompatibilidades y diferencias, por lo que el XNA Framework para la XBox 360 se denomina XNA Compact Framework. Por la misma razón, para realizar un videojuego para XBox tendremos que volver a compilar nuestro ensamblado para orientarlo a XBox 360.

Se está trabajando en la actualidad en un proyecto para portar el API de programación de XNA a otras plataformas del mismo modo que se hizo con .Net y el proyecto de software libre Mono. Por supuesto, no se hará sobre DirectX, pero el proyecto pretende hacer funcionar un API clónica a la de XNA sobre Mono, utilizando otras librerías nativas de sistemas de código libre como pueden ser SDL, OpenGL y otras. Muchas de estas librerías nativas ya tienen su propia traducción a API's soportadas por .Net a través de proyectos como el *Tao Framework*[22].

2.4.2. Arquitectura de XNA Framework

XNA Framework [53] se basa en la implementación nativa de *.NET Compact Framework*[23] para el desarrollo de la consola de **Microsoft Xbox 360** y **.NET Framework** en sus distintas versiones exclusivamente para el sistema operativo Windows. Incluye un amplio conjunto de bibliotecas de clases, específicos para el desarrollo de juegos, para promover la reutilización de código máximo a través de plataformas de destino. El marco se ejecuta en una versión de Motor de Lenguaje común, del inglés *Common Language Runtime* que se ha optimizado para proporcionar a los videojuegos un entorno de ejecución administrado. El tiempo de ejecución está disponible para las versiones de Windows **Windows XP** y **Windows Vista** y para la consola **Xbox 360** y la plataforma **Zune** (reproductor de audio digital de Microsoft que compite con el **IPod** de Apple). Los juegos en XNA están programados para el tiempo de ejecución y para que se ejecuten en cualquier plataforma

que admita el XNA Framework con mínima o ninguna modificación. Los juegos pueden escribirse en cualquier lenguaje compatible con .NET, pero oficialmente se admiten sólo C#.

XNA Framework encapsula a bajo nivel tecnológico los detalles relacionados en un juego de codificación, asegurándose de que el propio marco se encargue de la diferencia entre plataformas cuando los juegos son portados desde una plataforma compatible a otra, lo que permite que los desarrolladores de juegos puedan concentrarse más en el contenido y la experiencia de juego.

XNA Framework se integra con una serie de herramientas, tales como la plataforma múltiple de creación de audio *XACT*[24], para ayudar en la creación de contenido. XNA Framework proporciona apoyo para la creación de juego 2D y 3D y permite el uso de los controladores de Xbox 360 y vibraciones.

Para entender mejor todo esto, la siguiente figura el diagrama contempla todas las capas del framework y cómo se integran entre sí.

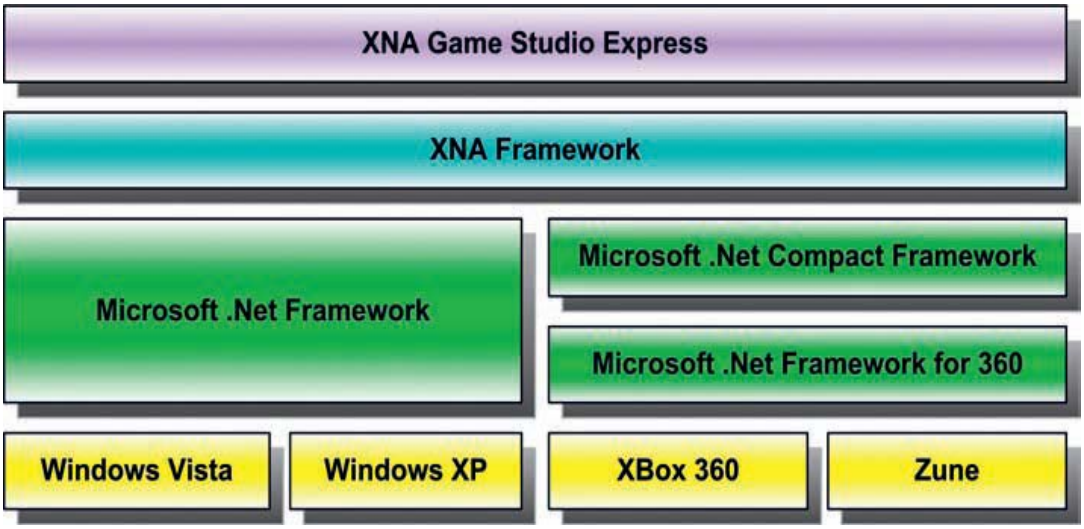

Figura 64. Diagrama de arquitectura del Framework de XNA

XNA Game Studio Express provee el XNA Framework, el cual permite la generación de aplicaciones dentro del concepto multiplataforma de Microsoft, es decir, Windows; XBox 360 y Zune como hemos visto antes. Para desarrollar aplicaciones basadas en XNA Framework podemos emplear productos que se pueden descargar gratuitamente, como **Visual C# Express Edition** junto con la versión 2.0 o 3.0 de XNA Framework.

XNA Game Studio Express extiende al lenguaje de programación C# para que ofrezca soporte al XNA Framework, el cual brinda un marco de trabajo para el desarrollo de juegos en las diferentes plataformas que soporta de acuerdo a su versión, en combinación con .Net; .Net Compact

Framework o su versión específica para Xbox 360. De esta manera, podemos aprovechar todos los conocimientos de C# y de .Net en el desarrollo de juegos o aplicaciones con fuerte utilización de la potencia de las Unidades de procesamiento de gráficos (GPU, del inglés *Graphics Processing Units*) modernas en 2D o en 3D. Por ejemplo, es posible combinar el acceso a bases de datos con la programación de juegos sin tener que cambiar de lenguaje de programación y de entorno de desarrollo.

Por otro lado, para entender las diferentes capas que componen el framework se muestra a continuación la figura 65.

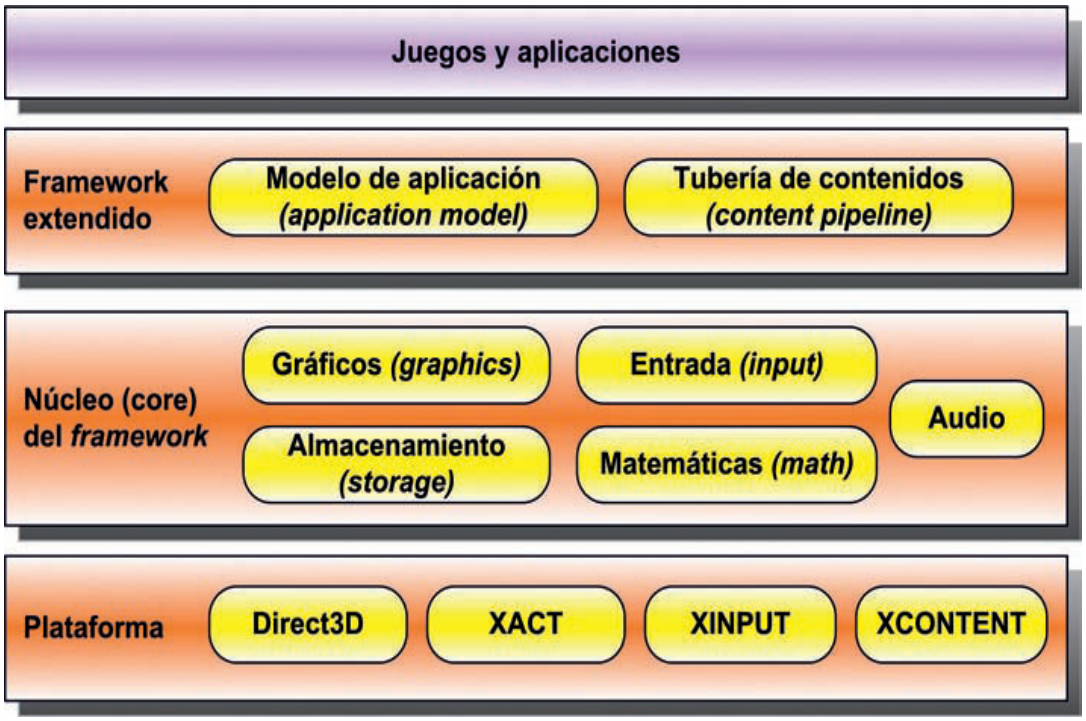

Figura 65. Capas del Framework de XNA

Como se puede observar, Direct3D está en la última capa, en la de plataforma. Encima de ésta se monta el núcleo (**core**) del framework, el framework extendido que contiene al modelo de aplicación y la tubería de contenidos y, sobre esto, aparece el juego o la aplicación, que es la parte que debe programar el desarrollador. XNA Framework provee al usuario de todas las necesidades para todas las actividades en común que presentan los juegos y las aplicaciones con uso de las capacidades de aceleración gráfica provistas por DirectX. En realidad, no se trata de nada novedoso, pues los principales productores de juegos y de esta clase de aplicaciones, en las diferentes plataformas en las cuales trabajan, siempre se generan sus propios frameworks de trabajo propietarios. La diferencia en este caso es que Microsoft intenta brindar mayores facilidades para todos los desarrolladores de juegos y de aplicaciones gráficas. Y, dado el avance que han tenido en los últimos años los entornos

de desarrollo, resultaba inadmisible que para comenzar a trabajar con DirectX se necesitará escribir tanto desde las bases y no contar con clases y estructuras predefinidas que faciliten las tareas comunes. La gran ventaja de XNA Framework es que, si no apuntamos a Xbox 360 inclusive tenemos herramientas gratuitas para comenzar a experimentar, lo cual, es muy beneficioso para el desarrollador.

Ahora bien, en la capa de aplicaciones y juegos, la cual es responsabilidad del desarrollador, debemos aportar el código que emplee los recursos provistos por el XNA Framework y los contenidos que alimentarán a la tubería de contenidos (C*ontent Pipeline*) de la que se hablará más adelante, como ser los modelos 3D, las texturas y los efectos, entre otras cosas. Para ello, se puede contar con componentes (*components*) que se generen dentro de la comunidad XNA, que tengan un comportamiento estandarizado y vayan ofreciendo determinadas funcionalidades no incluidas en XNA Framework pero las cuales pueden ser común a muchas clases de aplicaciones y, combinadas con un lenguaje de programación altamente flexible como lo es C#, nos vayan facilitando cada vez más el trabajo.

Las principales áreas en las cuales se divide el núcleo (core) del framework son las siguientes:

- **Gráficos (Graphics)**. Provee capacidades de carga de recursos y de transformación (*render*) de escenas de bajo nivel. Se encuentra montado sobre **Direct3D 9.0c**, pero simplifica radicalmente su control con respecto al trabajo directo con la API. Provee acceso a modelos, texturas, efectos y sombreadores (*shaders*) y permite trabajar tanto con gráficos 2D como 3D.
- **Audio**. Facilita clases y métodos en el espacio de nombres (*namespace*) *Microsoft.Xna.Framework.Audio* para reproducir ficheros de audio y sincronizarlos en el transcurso del juego. La reproducción de audio se lleva a cabo mediante la herramienta de creación de audio multiplataforma de Microsoft, conocida como XACT.
- **Input (Entrada)**. A través del espacio de nombres (namespace) *Microsoft.Xna.Framework.Input* provee clases y métodos para tomar las entradas que lleva a cabo el usuario a través de diferentes controles, como el teclado, el *gamepad*[25] y el ratón, entre otros. Simplifica muchísimo todo lo relacionado con la entrada para controlar el juego y las aplicaciones, sin entrar en la API DirectInput. Pues, está pensado teniendo en cuenta la temporización de esta clase de aplicaciones y las necesidades de sincronización de las entradas con los cuadros por segundo.

- **Math (Matemáticas)**. Provee clases y métodos para manipular vectores y matrices, facilitando las operaciones típicas que se deben llevar a cabo en los juegos con escenas 2D y 3D. Por ejemplo, provee mecanismos muy detallados para la detección de colisiones, uno de los algoritmos clásicos en el desarrollo de juegos.
- **Storage (Almacenamiento)**. Brinda clases en el espacio de nombres (namespace) *Microsoft.Xna.Framework.Storage* para permitir la lectura y escritura de ficheros relacionados con el estado de los juegos o con valores de parametrización de éstos.

Las dos áreas en las cuales se divide el framework extendido son las siguientes:

- **El modelo de aplicación (Application model)**. Provee un marco de trabajo para procesar la simulación del juego basada en intervalos de tiempo fijos (tiempo real) o variables. Facilita que arranquemos la aplicación codificando lo que está relacionado con su idea y no intentando resolver problemas propios de la plataforma. Por ejemplo, nos resuelve la manera de diseñar el lazo principal de control del juego, la creación y gestión de la ventana en donde se mostrarán los actores y varios patrones de mejores prácticas para solucionar los problemas más frecuentes. Su principal representante es la clase Game, dentro del espacio de nombres (namespace) *Microsoft.Xna.Framework*.
- **La tubería de contenidos (Content pipeline)**. Provee importadores y procesadores de contenidos para que los elementos artísticos (*art assets*) que se incluyen en el proyecto que conforma el juego se compongan de un modo determinado para que se puedan cargar en tiempo real en la plataforma en la cual se ejecute la aplicación (Windows, Xbox o Zune). De esta manera, el framework queda abierto a aceptar una amplia variedad de herramientas de creación de contenidos digital (DCC, del inglés *Digital Content Creation*), los cuales son aliados fundamentales de los desarrolladores y diseñadores de juegos y aplicaciones con un intenso contenido de recursos gráficos. Deja abierta la puerta para que se puedan incorporar importadores personalizados. La llamada más común para cargar estos contenidos en tiempo de ejecución es **ContentManager**.**Load**.

2.4.3. XNA Build

Xna Build es el conjunto de herramientas de gestión que ayudan al desarrollador de videojuegos a mantener, depurar y optimizar los recursos de los que dispone. Facilita al desarrollador a identificar las dependencias de los *pipelines* y proporciona acceso a las API's para habilitar más procesos de dependencia de datos. Esta dependencia de datos puede ser analizada con el objetivo de ayudar a reducir el tamaño de código de un juego; esto lo consigue buscando contenidos que no se suelen usar.

Uno de los pipelines más importantes de XNA es el **Content Pipeline (CPL)**. El CPL [54] es una API que permite a los desarrolladores y diseñadores incorporar contenidos multimedia en los proyectos creados con XNA framework como por ejemplo imágenes, sonidos, contenido 3d, efectos, etc.

Por defecto el CPL soporta una amplia gama de formatos de archivo diferentes los cuales son usualmente usados en la industria de los videojuegos, el CPL facilita el acceso a estos archivos y proporciona un interfaz de acceso unificado que permite acceder a dichos recursos desde cualquier objeto utilizado dentro del juego sin necesidad de hacer uso de múltiples referencias cruzadas lo cual va en favor de la independencia de cada componente, cosa que en un juego generalmente es una tarea muy difícil de hacer. Sin embargo las capacidades del CPL podrían verse limitadas por la cantidad de archivos que soporta puesto que es muy común que en la industria de los videojuegos se usen formatos de archivo independientes de acuerdo a las necesidades particulares de cada proyecto, es aquí donde reside una de las más importantes características del CPL, que es **extensible**.

El CPL incorpora un marco de trabajo que permite fácilmente incorporar soporte a diferentes tipos de archivo e incluso extender la funcionalidad de un tipo de archivo ya soportado.

El CPL trabaja sobre tres espacios de nombres, cada uno de ellos con sus propios elementos que son los siguientes:

- Namespace: **Microsoft.Xna.Framework.Content.Pipeline**
  Elementos: ContentImporter, ContentProcessor
- Namespace: **Microsoft.Xna.Framework.Content**
  Elementos: ContentTypeReader
- Namespace:**Microsoft.Xna.Framework.Content.Pipeline.Serialization.Compile**
  Elementos: ContentTypeWriter

El *ContentImporter* desencadena el proceso y es el encargado de realizar por un lado la lectura física del archivo que se va a importar y por otro lado se va a encargar de colocar la información en un objeto capaz de almacenarla. A continuación, la información colocada en el objeto es pasada a través de un *ContentProcessor* el cual se encarga de realizar transformaciones en la información cargada en el paso anterior. Por último, el *ContentTypeWriter* se encarga de escribir ese objeto en un archivo con el formato del CPL. Este proceso se lleva a cabo en tiempo de compilación como muestra la figura 66.

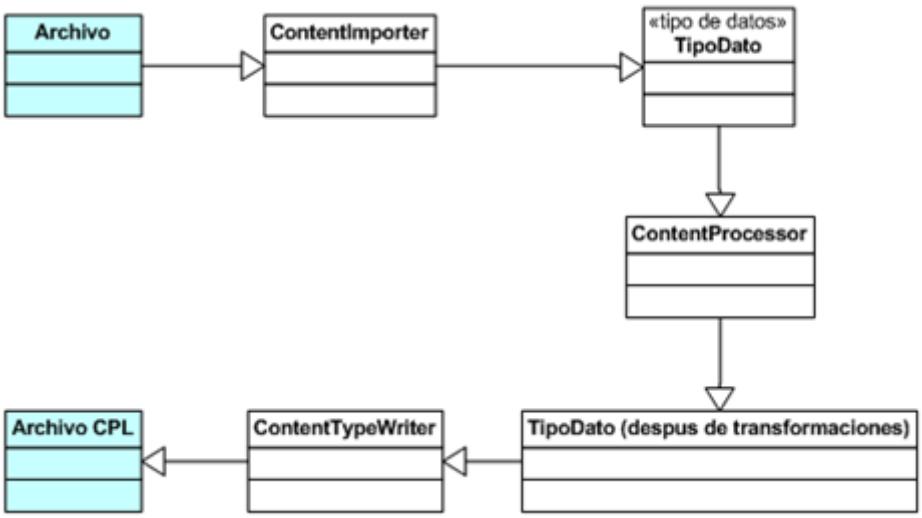

Figura 66. Content Pipeline en tiempo de compilación

A continuación, ya en tiempo de ejecución cuando el juego ya está en curso, el *ContentTypeReader* lee los archivos creados por el CPL para cargar en memoria el recurso almacenado. Debido a que se debe adicionar una referencia a estas clases en el CPL es necesario que los tres primeros objetos existan compilados en una librería dinámica, mientras que el último, el juego puede hacer una parte directamente. La figura 67 muestra el proceso en tiempo de ejecución.

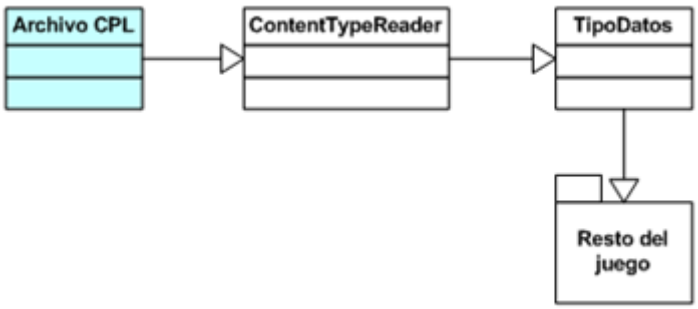

Figura 67. Content Pipeline en tiempo de ejecución

2.4.4. XNA Game Studio

XNA Game Studio [55] es el Entorno de Desarrollo Integrado (IDE) que se usa para desarrollar juegos en XNA. Existen varias versiones para usuarios diseñadas según las necesidades de éstos. Las características principales de las versiones de Game Studio son las siguientes:

- **Game Studio 2.0.** Diseñado para ser utilizado en Visual Studio 2005 ofrece una API de red usando Xbox Live (Servicio en línea de Microsoft) en Windows y Xbox 360.
- **Game Studio 3.0.** Puede ser usado en Visual Studio 2008 y la principal característica es que permite producir juegos de la plataforma Zune. Es compatible con C# 3.0 y LINQ. Otra característica importante es un nuevo modo de prueba para que el desarrollador pueda hacer un seguimiento más eficaz en las pruebas del juego.
- **Game Studio 3.1.** La principal novedad en esta secuela es que se incluyó el soporte para la reproducción de video (antes inexistente) y una API revisada para el audio.
- **Game Studio 4.0.** Es la última versión en la actualidad y está destinada al desarrollo de juegos para *Windows Phone 7 Series*, además también incluye mejores características en el desarrollo de juegos para la Xbox 360 y PC. La integración con Visual Studio 2010 permite a los desarrolladores construir un sólo proyecto que, a continuación, realizando pequeñas modificaciones, se pueda ejecutar en cada plataforma. Además, una de las características más destacables es que incluye una API de aceleración 3D por hardware para Windows Phone 7 Series.
- **Game Studio Express.** Esta versión está destinada a los estudiantes y aficionados y está disponible como descarga gratuita. Express proporciona un kit de principiantes para el rápido desarrollo de los géneros específicos de juegos, tales como la plataforma o estrategia en tiempo real entre otras. Los desarrolladores pueden crear juegos de Windows de forma gratuita con XNA Framework, pero para ejecutar sus juegos en la Xbox 360 hay que pagar una cuota anual para la admisión al **Club de creadores de XNA**.
- **Game Studio Professional.** Basado en *Visual Studio 2005 Team System*, proporciona una estructura para la colaboración entre los creadores de contenido, los programadores, administración y evaluadores. Las tareas de administración del proyecto, tales como la gestión de activos, seguimiento de defectos,

automatización de proyectos y listas de artículos de trabajo, son automatizadas por XNA Studio.

2.5. Otras arquitecturas

Actualmente existe gran cantidad de arquitecturas en el mercado, tanto libres como de pago, orientados a las distintas plataformas de juego y sistemas operativos existentes *Windows*, *Linux* y *Mac OS X*. Este tipo de plataformas proporcionan una serie de funcionalidades que incluyen un motor de renderización o "*render*" para gráficos 2D y 3D, un motor de física de cuerpos también conocido como motor de colisiones o motor de detección de colisiones, soporte para audio y sonidos, scripts, animaciones, inteligencia artificial, juego en red, streaming de contenidos, manejo de memoria, multijugador etc.

Con el tiempo estas plataformas se han convertido en verdaderos motores, reutilizándose en múltiples videojuegos.

La historia de estos motores es larga, algunos piensan que comenzó con la Atari 2600 y su motor de desarrollo conocido por muchos hoy día como "*kernel*" (dando a entender que fue la primera herramienta que se puede considerar núcleo o motor de una serie de videojuegos). La primera generación de motores estuvo dominada por tres que hoy día siguen evolucionando y son base de juegos muy importantes. Estos son ***BRender*** de *Argonaut Software*, ***Renderware*** de *Criterion Software Limited* y ***Reality Lab*** de RenderMorphics. *Reality Lab* era el más rápido de los tres, por lo que poco después de aparecer fue comprado por *Microsoft* y utilizado como base para ***Direct3D***. ***Renderware*** fue adquirido más tarde por *EA*, siendo antes de esta adquisición motor de juegos tan importantes como los de las sagas *Grand Theft Auto* y *Burnout*.

La evolución del mundo de los videojuegos es tal, que ha hecho que hoy día existan motores similares a las versiones comentadas orientadas a desarrolladores amateur y que proporcionan muchas de las características de los motores comerciales. A continuación se hace referencia a los más importantes:

- **Adventure Game Studio (AGS)** [56]**.** Plataforma de creación de aventuras gráficas. Proporciona una interfaz gráfica desde la que es posible crear la aventura gráfica sin ningún tipo de conocimiento de programación. Para desarrolladores más avanzados incluye un editor de scripting. Se distribuye en versión gratuita. A partir de la versión 2.72 el lenguaje de script está basado en un lenguaje orientado a objetos. Una vez dominado el lenguaje de script las

limitaciones del programa son muy pocas e incluso se pueden hacer juegos de otros estilos, como *RPG*. La nueva versión 3.0, usa *Visual Studio .NET* sustituyendo al scripting. Una de sus desventajas es que, hasta la versión 3.0, los juegos desarrollados consumen muchos recursos, pero con la conversión a *VS .NET* se espera solucionar este problema y potenciar la herramienta.

- **Build** [57]**.** Es un motor para juegos de disparo en primera persona tipo *Doom*. Creado por *Ken Silverman* para *3D Realms*, representa todos sus elementos en una malla 2D, utilizando formas 2D y sprites planos. Se considera un motor 2.5D, ya que a la geometría básica 2D le añade la componente de la altura. La herramienta, escrita en *C*, fue liberada en el año 2000.
- **Blender 3D** [58]**.** Es una aplicación de diseño 3D. Es gratuita y se distribuye bajo la licencia *GNU General Public License*. Su funcionalidad principal es la de diseño 3D de figuras y modelos, pero también es posible realizar simulaciones de fluidos, detección de colisiones o simulaciones físicas gracias por lo que en sus últimas versiones la potencia del motor se ha aprovechado para crear aplicaciones interactivas 3D incluidos juegos. Es compatible con *Linux*, *Mac OS X*, y *Microsoft Windows*. Todo el diseño se realiza a partir de su interfaz gráfica pero además incluye soporte de script en *Phyton*. Sus detractores critican la complejidad de su interfaz pero lo cierto es que es una herramienta muy potente tanto gráficamente como en físicas y colisiones y en su última versión, la 2.5, se ha proporcionado una nueva interfaz, y nuevos sistemas de animación y de entrada de usuario que aumentan la potencia de la herramienta.
- **Crystal Space** [59]**.** Framework para el desarrollo de aplicaciones 3D. Se suele usar como motor de videojuegos, pero el framework es más general y puede ser usado para cualquier tipo de diseño 3D. Diseñado para ser totalmente portable, se ejecuta en *Microsoft Windows*, *Linux* y *Mac OS X*. *Crystal Space* es gratuito y se distribuye bajo la licencia *LGPL*[26]. Es compatible con *OpenGL*, *SDL*, *X11* y *SVGALib*. Está programado en *C++* usando un diseño orientado a objetos.
- **Dim3** [60]**.** También conocido como ***Dimension3***, es una herramienta de código libre orientada al diseño 3D y desarrollo de videojuegos. La herramienta es compatible con *Mac OS X*, *Microsoft Windows* y *Linux*. *dim3* se apoya en otra serie de herramientas y estándares para proporcionar sus funcionalidades, utiliza *OpenGL* para el renderizado, *OpenAL* para el audio, *JavaScript* como lenguaje de scripting, *XML* como formato de datos y *Simple DirectMedia Layer* como gestor de ventanas y video. Su primera versión estable es de **2007** pero ya ha

conseguido el apoyo de *Apple* y se considera una de las herramientas que más crecerá en los próximos años.

- **Doom Engine** [61]**.** Es el motor gráfico que se usó para los videojuegos *Doom* y *Doom II*. Originalmente desarrollado en computadoras *NeXT*, fue portado a *DOS* y posteriormente a otras consolas y sistemas operativos. El código fuente se hizo público en 1997 bajo licencia GNU en 1999. No es un verdadero motor 3D, ya que no hay componente de altura. Está algo obsoleto aunque se sigue usando para desarrollar clones de *Doom*.
- **Gamebryo** [62]**.** Es un motor de gráficos 3D escrito en C++ y dirigido además al desarrollo de videojuegos. Su gran ventaja es que da soporte a todas las plataformas existentes actualmente *Windows DirectX 9* y *DirectX 10*, *Wii* / *WiiWare*, *PlayStation 3*, *PSP* y *XBOX 360* (incluyendo *XBOX Live Arcade*). Su principal inconveniente es que es de pago.
- **3D Game Studio** [63]**.** También conocido como *3D Game Studio*, es una aplicación muy conocida para el desarrollo de juegos en 2D y 3D. El núcleo del programa se denomina A7, un motor gráfico que controla la imagen y el comportamiento del mundo virtual desarrollado. Es muy potente y trabaja de igual forma con espacios interiores y exteriores. También soporta sombras estáticas y dinámicas. A su motor 3D se le unen otros que complementan el programa y que permiten la producción de videojuegos con resultados muy espectaculares: motor de efectos y partículas, motor de física y colisiones, motor bidimensional, motor sonoro, motor de red, etc. La aplicación permite desarrollar un juego sin tener conocimientos de programación, pero para desarrolladores más expertos proporciona *C-Script* (también conocido como *Lite-C*), una versión simplificada de *C++*. Existen cuatro versiones y todas tienen actualizaciones gratuitas.
- **Irrlicht** [64]**.** Es un motor 3D gratuito y de código abierto, escrito en *C++*, que puede ser usado tanto en *C++* como con lenguajes *.NET*. Sus características principales incluyen compatibilidad *Windows*, *Linux* y *MacOS*, soporte *Pixel Shader* y *Vertex Shader*, capacidad de manejo de interiores y exteriores, sistema de animaciones *skeletal*[27] y *morph*[28], sartículas, mapas de luces *enviromment mapping*[29] y sombras *stencil buffer*[30], sistema para interfaces 2D, rápido y fácil sistema de colisiones, infinidad de formatos 3D, texturas y lenguajes de programación soportados, así como un API totalmente documentada con ejemplos y tutoriales. Es una herramienta muy potente pero debido a que está orientada al desarrollo desde código es poco conocida.

- **M.U.G.E.N.** [65]**.** Es un motor gráfico para crear juegos de lucha en 2D. Ha tenido mucho éxito entre los aficionados y hay una gran comunidad de usuarios que lo apoya debido a que con él, se pueden crear personajes, escenarios y barras de vida a partir de juegos existentes. Permite crear juegos a partir de personajes de distintos juegos y compañías; se puede desde simplemente añadir personajes de cualquier juego (que sean compatibles con el sistema M.U.G.E.N.), escenarios varios, paquetes de visualización de pantalla (llamados screenpacks) o barras de vida; hasta crear un juego de peleas completo con todo un sistema establecido, personalizado y modificado al gusto del creador. Lo que más destacan sus usuarios es la capacidad de crear juegos con personajes de cualquier otro juego del que se pueda sacar los sprites y programar sus movimientos.
- **Quest3D** [66]**.** Es la fusión de un motor de videojuego con una plataforma de desarrollo. Se usa para arquitectura, diseño de producto, videojuegos, software de entrenamiento y simuladores. Los datos y animaciones son importados de paquetes *CAD* a *Quest3D* donde son utilizados para la creación de aplicaciones interactivas 3D en tiempo real. Una de las características más importantes de *Quest3D* es que su entorno de desarrollo, pudiéndose modificar la aplicación mientras esta se ejecuta, pero sin la posibilidad de compilación de código como en los entornos de programación habituales. Este tipo de desarrollo lo hace accesible a más usuarios pero también lo limita con respecto a los más expertos. Existen varios tipos de licencias y las aplicaciones finalizadas pueden ser publicadas en diferentes formatos: fichero ejecutable "*.exe*" y visor WEB basado en control *ActiveX*.
- **RPG Maker** [67]**.** Herramienta para *Windows* que permite al usuario crear sus propios videojuegos de rol. Incluye un editor de mapas, un editor de eventos y un editor de combates. Todas las versiones de *RPG Maker* necesitan el *RTP* (*Run Time Package*) que incluye gráficos para mapeados, personajes, música, efectos de sonido, etc. que pueden ser utilizados para crear nuevos juegos. Una característica muy destacada por los usuarios es que en las versiones de PC, es posible agregar nuevos materiales gráficos y sonoros personalizados a las librerías de proyecto. Hay una gran comunidad de usuarios que comparten librerías de gráficos, sonidos, fondos y demás archivos que ayudan a la elaboración de una nueva creación, o la modificación de una ya creada anteriormente. Como inconveniente a destacar, que sólo está orientado a juegos de rol.

- **Source** [68]**.** Es un motor de videojuego desarrollado por la empresa *Valve* para las plataformas *Windows*, *XBOX*, *XBOX 360* y *PlayStation 3*. Utilizado por primera vez con el videojuego *Counter-Strike: Source* y después con *Half-Life 2*. *Source* fue creado para ir evolucionando poco a poco mientras la tecnología avanza, al contrario que los cambios de versión bruscos de sus competidores. Esto se vuelve especialmente relevante cuando se considera que está ligado a *Steam*, el cual baja las actualizaciones automáticamente lo que hace que nuevas versiones del motor puedan llegar a toda la base de usuarios instantáneamente. Es muy potente en cuanto al manejo de animaciones, modelos 3D, físicas y colisiones. Está orientado a desarrolladores comerciales y sus licencias son de pago, aunque existen licencias especiales para *modders*[31].

Estas son sólo algunas de las herramientas y plataformas que es posible encontrar en la red y que permiten diseñar y desarrollar videojuegos. Muchos de ellos tienen el inconveniente de las licencias, sobre todo los más avanzados y potentes. Otro añaden el problema del desarrollo, resultando muy complejo en algunos casos en los que la única manera de desarrollar es mediante código; o muy simple si se limitan a proporcionar un interfaz y no dar soporte adicional de código. De la misma manera muchas de estas herramientas se apoyan en librerías externas, o lenguajes de programación que les permitan proporcionar funcionalidades adicionales, sobre todo las que se centran en las 3D, de forma que es necesario manejar varias librerías, herramientas y lenguajes para conseguir un resultado óptimo. Por último resaltar que la mayoría proporciona una mínima documentación y escasos ejemplos, lo que hace aún más complicado el desarrollo.

Casi todos estos problemas quedan solventados en mayor o menor medida en *XNA*, API utilizado para el desarrollo de este proyecto.

Capítulo 3

Gestión del proyecto

En este capítulo se explican las fases por las que ha transcurrido el proyecto.

En la fase de Análisis se explica de dónde surge la idea del proyecto, la identificación de requisitos, los casos de uso y los diagramas de actividad y secuencia.

En la fase de Diseño se mostrará el diagrama de clases y se explicarán todas ellas.

La fase de Implementación explica todos los elementos, algoritmos y cuestiones tenidas en cuenta para desarrollar el juego.

3.1. Fase de Análisis

La fase de análisis es la primera fase del ciclo de vida de un proyecto y es fundamental en cualquier elaboración que contenga una arquitectura. Se parte de una necesidad o una idea y a partir de esta se toman las decisiones necesarias para llevar a la práctica el proyecto.

En un proyecto real, lo normal es que haya un equipo de trabajo formado por un jefe de proyecto, analistas, desarrolladores, diseñadores, etc., pero en este caso todos estos roles los elabora la misma persona.

En primer lugar se va a plantear qué tipo de juego se va a realizar y por qué, a continuación comienza la fase de análisis especificándose los requisitos (menús, opciones, movimientos del jugador, número de niveles, elementos que lo componen, etc.). Otra parte importante del análisis, después de modelar el concepto en los distintos objetos que se puede descomponer, consiste en analizar los estados en los que se pueden encontrar estos y cómo transitar de unos a otros. Para ello se usarán técnicas de modelado como *UML*[32], para la elaboración de *diagramas de casos de uso*[33].

Una vez esté analizada y definida la idea se pasará al siguiente punto, la fase de diseño, que abarcará cada elemento por separado de la estructura y lo acercará más al desarrollador; aunque sin llegar todavía a la implementación.

3.1.1. Idea inicial

La idea original de este PFC se basaba en la realización de un juego basado en la arquitectura XNA y que abordase todos los requisitos y fases por los que transcurre un videojuego comercial.

En este sentido, se apostó por la elaboración de un clon de uno de los juegos arcade más famosos y que más adeptos tiene en el mundo, como es el Bubble shooter. Se eligió este juego ya que a pesar de ser un juego sencillo de entender y de jugar, ofrece desde el punto de vista del desarrollador un amplio abanico de posibilidades para adaptarlo a las necesidades que se pedían como son la implementación básica de los personajes (en este caso las bolas), dualidad en los controles (teclado y ratón), tratamiento de las colisiones, implementación de niveles, elaboración de un sistema de puntuación dinámico y navegación por pantallas. Desde el punto de vista del jugador, ofrece los factores imprescindibles para que guste como son un fácil aprendizaje, capta la atención del jugador desde el primer momento ya que aunque es fácil

conseguir hacer puntos, a medida que transcurre el tiempo, el mapa se va haciendo más complicado y obliga al jugador a tener habilidad mental y mecánica para hacer más puntos; por tanto, también motiva al jugador a superarse a sí mismo. Aunque se trata de un juego para un solo jugador, también se puede jugar en compañía, ofreciendo retos entre los jugadores para saber quién tiene más capacidad de destruir bolas antes de que el mapa se llene de ellas.

El guión gráfico no supuso realmente una gran complejidad, ya que la base del juego son las bolas de colores, y la mecánica del juego no ofrece más objetos externos que éstos, sin embargo, se apostó por intentar realizar un contenido gráfico sencillo para que el jugador sólo tenga que centrarse en el objetivo del juego pero que a la vez resultase agradable para poder jugar todo el tiempo deseado sin acabar visualmente cansado.

3.1.2. Identificación de requisitos

El propósito de este punto consiste en la identificación de los **requisitos de usuario** y los **requisitos software**. Una descripción del sistema a desarrollar a nivel de capacidades, restricciones, características del usuario, entorno operacional y dependencias.

Los requisitos de usuario reflejan las necesidades que debe cubrir el sistema desde el punto de vista del usuario. A partir de estos requisitos se podrá entender lo que espera el usuario del sistema a construir y llevarlo a cabo con éxito. Dentro de los requisitos de usuario se distinguen dos tipos: **requisitos de capacidad**, que representan una necesidad o servicio requerida por el usuario, y **requisitos de restricción**, que son las restricciones impuestas por los usuarios sobre cómo se debe resolver el problema o cómo se debe alcanzar un objetivo.

Dentro de los requisitos software, se encuentra los **requisitos funcionales** que describen lo que debe hacer el sistema, y los **requisitos de rendimiento**, **interfaz**, **operación**, **recursos**, **comprobación**, **documentación**, **seguridad**, **calidad**, **mantenimiento**, **daño** y **aceptación de pruebas** que limitan la forma en que debe llevarse a cabo.

Estos requisitos tienen un alto grado de importancia en cuanto a verificación y seguimiento del juego, ya que este deberá cubrir todos y cada uno de los requisitos identificados para afirmar que está completo.

Los requisitos de capacidad (RUC) de este proyecto son los siguientes:

- **RUC-01** *Ejecutable*: Lanzar el juego a través de un ejecutable.
- **RUC-02** *Jugar*: Comenzar una nueva partida.
- **RUC-03** *Configurar Opciones*: Configurar los parámetros ajustables del juego. Los parámetros configurables son:
    - Seleccionar nivel de dificultad del juego.
    - Seleccionar nivel de dificultad de las puntuaciones.
- **RUC-04** *Activar los valores por defecto*: Aplicar los valores por defecto a los parámetros ajustables del juego.
- **RUC-05** *Guardar opciones*: Guardar los valores seleccionados para los parámetros ajustables del juego. Los valores seleccionados son guardados durante la ejecución del juego.
- **RUC-06** *Mostrar puntuaciones*: Mostrar las puntuaciones más altas por nivel alcanzadas en el juego.
- **RUC-07** *Salir*: Cerrar la ventana de juego. La opción de salir es aplicable también al retorno de pantallas.
- **RUC-08** *Mover el personaje*: Desplazar el lanzador de bolas de izquierda a derecha hasta posicionarse en el ángulo deseado para disparar.
- **RUC-09** *Manejar bolas*: El lanzador dispara la bola del color que indique en el momento para hacerla colisionar con otra que esté en el mapa, intentando que la bola con la que colisione sea del mismo color que la que se lanza.
- **RUC-10.** *Pausa*: Detener una partida temporalmente manteniendo su estado actual para reanudarla más tarde sin cerrar el juego.
- **RUC-11.** *Puntuación máxima*: El jugador podrá introducir un nombre de usuario para registrar su puntuación y así identificarse en la lista de puntuaciones máximas en el nivel que se ha jugado.

A continuación, se mostrarán los requisitos de restricción impuestas por el usuario (RUR):

- **RUR-01.** *Dificultad*: El jugador podrá seleccionar tres niveles de dificultad: Fácil, Medio y Difícil.
- **RUR-02.** *Disparar bola*: Para disparar una bola, previamente el jugador ha seleccionado la dirección del ángulo donde quiere disparar. El disparo se realiza mediante el teclado con la tecla de ESPACIO o con el ratón pulsando el botón izquierdo.

- **RUR-03.** *Puntuación máxima*: Se mostrarán las diez puntuaciones más altas dentro de cada nivel.
- **RUR-04.** *Nombre del jugador*: El jugador que haya conseguido una puntuación dentro de las diez mejores del nivel jugado, podrá introducir un nombre de usuario de hasta diez caracteres siendo válidas las letras del abecedario, los número del 0 al 9 y la barra espaciadora.
- **RUR-05.** *Dificultad nivel fácil*: Los parámetros para establecer el nivel fácil son un tiempo de 10 segundos a la generación aleatoria del mapa de bolas para dar la posibilidad de tener más bolas del mismo color agrupadas y un tiempo de 40 segundos entre la aparición de una nueva fila de bolas en el mapa hasta la siguiente.
- **RUR-06.** *Dificultad nivel medio*: Los parámetros para establecer el nivel medio son un tiempo de 20 segundos a la generación aleatoria del mapa de bolas para dar la posibilidad de tener un nivel intermedio de bolas del mismo color agrupadas y un tiempo de 30 segundos entre la aparición de una nueva fila de bolas en el mapa hasta la siguiente.
- **RUR-07.** *Dificultad nivel difícil*: Los parámetros para establecer el nivel difícil son un tiempo de 30 segundos a la generación aleatoria del mapa de bolas para dar la posibilidad de tener menos bolas del mismo color agrupadas y un tiempo de 20 segundos entre la aparición de una nueva fila de bolas en el mapa hasta la siguiente.
- **RUR-08.** *Ayuda*: La ayuda consta de una pantalla donde se muestran las características principales del juego tanto de reglas como de opciones de juego.
- **RUR-09.** *Formatos*: El formato de los ficheros de sprites debe ser .JPG o .PNG y el formato de los ficheros de audio para efectos sonoros .WAV. Los archivos de puntuaciones máximas son ficheros XML.
- **RUR-10.** *Entorno operativo*: El juego compilado será compatible con los sistemas operativos Windows XP, Windows Vista y Windows 7 que tengan instalado XNA Framework Redistributable 3.1.

Los requisitos de software funcionales (RSF) recopilados son los siguientes:

- **RSF-01.** *Lanzar el juego*: El juego se lanzará mediante la ejecución de un fichero .EXE que será generado al compilarlo en un PC.
- **RSF-02.** *Mostrar Menú Principal*: Una vez lanzado el juego se mostrará la pantalla de Menú Principal que incluye el logo del juego y las siguientes entradas:

- Jugar
- Nivel de dificultad
- Mejores puntuaciones
- Ayuda
- Salir

- **RSF-03.** *Mostrar Menú Nivel de dificultad*: Para mostrar este Menú hay que seleccionar la entrada correspondiente. Dicho menú muestra las siguientes entradas con sus respectivos valores a elegir:
  1: Fácil
  2: Medio
  3: Difícil
- **RSF-04.** *Valores por defecto*: El valor por defecto es el referente al nivel de dificultad, y si el jugador no selecciona un nivel de forma manual, por defecto se aplica el nivel fácil.
- **RSF-05.** *Salir y guardar*: Es posible salir de cada menú volviendo al menú principal pulsando la tecla **ESC**. Durante el juego, si se pulsa esta tecla el juego permanecerá en pausa permitiendo al jugador poder continuar por donde se había quedado o salir del juego. En el menú de Nivel de Dificultad, se puede salir presionando ESC o alguna de las opciones de nivel.
- **RSF-06.** *Puntuaciones*: Desde el Menú Principal se podrán mostrar las diez puntuaciones máximas seleccionando la opción **Mejores Puntuaciones**. El Menú Puntuaciones muestra el nombre y la puntuación alcanzada dentro del nivel seleccionado.
- **RSF-07.** *Salir del juego*: Desde el Menú Principal el jugador podrá cerrar la ventana de juego seleccionando la opción **Salir**. Adicionalmente se podrá salir mientras el jugador esté jugando pulsando la tecla **ESC** y eligiendo la opción **Salir**.
- **RSF-08.** *Jugar partida*: Desde el Menú Principal el jugador podrá comenzar una nueva partida seleccionando la opción **Jugar**.
- **RSF-09.** *Ayuda*: El usuario puede obtener una ayuda sobre cómo jugar seleccionando la opción **Ayuda** dentro del menú principal.
- **RSF-10.** *Pausar partida*: Una vez iniciada la partida el jugador puede pausarla en cualquier momento pulsando **ESC**. Desde la pantalla de Pausa el jugador puede reanudar la partida seleccionando **Continuar**, o abandonar la partida seleccionando **Salir**.

- **RSF-11.** *Controlar el lanzador*: El jugador puede desplazar el lanzador horizontalmente. El desplazamiento responde a las flechas izquierda y derecha del teclado y al movimiento de izquierda a derecha del ratón. El movimiento genera un ángulo de disparo señalado por una flecha.
- **RSF-12.** *Disparar bolas*: El jugador puede disparar con la barra espaciadora o con el botón izquierdo del ratón una vez haya situado el lanzador en el ángulo deseado. No se pueden disparar bolas simultáneamente, por lo que sólo se puede disparar una nueva bola, una vez que la anterior haya colisionado con otra bola o haya explotado.
- **RSF-13.** *Explotar bolas*: Una bola disparada y que colisione con 2 ó más bolas del mismo color que estén agrupadas producirá una explosión de todas ellas y desaparecerán del mapa de bolas.
- **RSF-14.** *Subir de nivel*: Cuando el jugador supera los múltiplos de 1000 puntos avanza de nivel dentro del juego y el tiempo entre la generación de nuevas filas disminuye 1 segundo por cada nivel alcanzado.
- **RSF-15.** *Fin de partida*: El mapa inicial de bolas consta de 8 filas. Cuando el mapa llega a la fila 17 se superará la línea que limita el mapa con el lanzador y el juego finalizará. Si la puntuación obtenida está dentro de las 10 mejores del nivel en el que se ha jugado, el jugador podrá escribir su nombre en la lista de mejores puntuaciones, si no, aparecerá la pantalla de final de juego dando al jugador la opción de volver al menú principal o salir del juego.
- **RSF-16.** *Ganar*: Realmente este juego no tiene una meta determinada, sino que el objetivo es explotar el máximo número de bolas posibles para conseguir el máximo número de puntos posibles antes de que el mapa de bolas supere las 17 filas.
- **RSF-17.** *Puntuaciones*: La forma de conseguir puntos es básicamente explotar el máximo número de bolas agrupadas del mismo color, siendo el sistema de puntuación el siguiente:
    - 1 bola = 10 puntos, siendo 3 bolas el mínimo exigido y por tanto 30 puntos.
    - Por cada bola adicional superando las 3 bolas se sigue la fórmula siguiente:

      (Numero de Bola / 2) x 10;
    - Por tanto, si se explotan 7 bolas juntas, los puntos conseguidos serían **(3x10) + (4/2)x10 + (5/2)x10 + (6/2)x10 + (7/2)x10**. En total **130 puntos**.

- Por cada nivel superado, se asignan puntos, siendo éstos el número de nivel superado multiplicado por diez, es decir, si se supera el nivel 3, se asginan **3x10 = 30 puntos.**
- Adicionalmente, si el jugador es capaz de limpiar el mapa de bolas y quedarse sin bolas en la pantalla, se le compensará con **10000** puntos extra.

Una vez enumerados los requisitos software funcionales se exponen los requisitos software de interfaz (RSI):

- **RSI-01.** *Nombre del jugador*: El nombre del jugador sirve para identificar a los jugadores con puntuaciones máximas.
- **RSI-02.** *Formato de puntuaciones máximas*: El Menú Puntuaciones mostrará las diez puntuaciones máximas de cada nivel de dificultad con el formato Nombre, Puntuación.
- **RSI-03.** *Formato de archivos*: Los archivos de sprites incluidos en el proyecto serán del formato .JPG o .PNG y los efectos de sonido en formato .WAV. Los archivos de puntuaciones máximas son ficheros XML.
- **RSI-04.** *Niveles*: Los niveles dentro del juego se alcanzan cuando el jugador supera múltiplos de 1000 puntos, consiguiendo de este modo sumar puntos. Por cada nivel superado, se asignan el número de nivel superado multiplicado por cien, es decir, si se supera el nivel 3, se asginan **3x10 = 30 puntos.**
- **RSI-05.** *Software y formato de ficheros*: Se utilizará Visual Studio .NET 2008 y XNA 3.1 como herramientas de desarrollo. El lenguaje de programación usado será C#. Los ficheros que almacenan las puntuaciones estarán en formato XML.
- **RSI-06.** *Tamaño*: Una vez finalizado el desarrollo, el juego, incluyendo todo el contenido de animaciones y efectos sonoros no excederá de 50Mb, máximo tamaño de los juegos con valor de 200 *MPoints.*
- **RSI-07.** *Desarrollo*: Para el desarrollo del juego es necesario un equipo con Windows XP, Windows Vista o Windows 7 con las siguientes características:
  - Visual Studio .NET 2008
  - XNA Game Studio 3.1
  - Una tarjeta gráfica que soporte shaders 1.1 (recomendado 2.0)
- **RSI-08.** *Ejecución*: Para la ejecución del juego es necesario un equipo con las siguientes características:
  - Windows XP, Windows Vista, Windows 7.
  - XNA Framework Redistributable 3.1.

- DirectX instalado (8.1 o superior, preferiblemente 9.0c o superior)
- Una tarjeta gráfica que soporte shaders 1.1 (recomendado 2.0)[34]

Hasta aquí el apartado destinado a la identificación de requisitos. No se han identificado requisitos esenciales de comprobación ni seguridad. En cuanto a los requisitos de documentación y calidad vienen marcados por la exigencia propia de un proyecto fin de carrera por lo que tampoco se reflejan requisitos adicionales de este tipo.

3.1.3. Especificación de casos de uso

Una vez expuestos los requisitos de la aplicación, se deben elaborar los distintos diagramas de casos de uso resultantes de los requisitos definidos, así como la especificación textual de cada caso de uso para su mejor entendimiento. A diferencia de otras aplicaciones, los videojuegos, permiten agrupar los casos de uso en varios grupos comunes a casi todos los videojuegos. De este modo, se tienen los casos de uso de tipo ***Player Input***, que agrupan las funcionalidades relacionadas con la entrada de usuario, los de tipo ***View*** o ***Display***, que agrupan las funcionalidades relacionadas con menús y gráficos; las de tipo ***Game Object Interaction***, que recogen la interacción del juego con los distintos objetos que lo componen y por último las de tipo ***Miscellaneous***, que recogen acciones variadas que no se han encajado en los otros grupos.

En los casos de uso se especifica qué hace el sistema en respuesta a una interacción de un usuario externo, por tanto no se tendrá en cuenta al propio sistema como actor (sólo sistemas externos que interactúen con el propio se pueden considerar actores). De esta forma, y por la naturaleza propia de algunas funcionalidades típicas de los videojuegos, surgirán casos de uso que quizá parezcan propios del sistema como la detección de colisiones; pero serán representados con el jugador como actor.

Como se ve en la figura 68, los casos de uso quedarían agrupados de la siguiente manera:

En **Player Input**: Jugar, Pausar partida, Disparar, Salir, Mover lanzador a la izquierda y Mover lanzador a la derecha.

En **View/Display**: Mostrar Menú, Mostrar Ayuda y Mostrar Máximas puntuaciones.

En **Game Object Interaction**: Colisionar bola con bola, Destruir bolas y Colisionar bola con ventana.

En **Miscellaneus Actions**: Seleccionar nivel de dificultad, Introducir nombre y Seleccionar nivel de máximas puntuaciones.

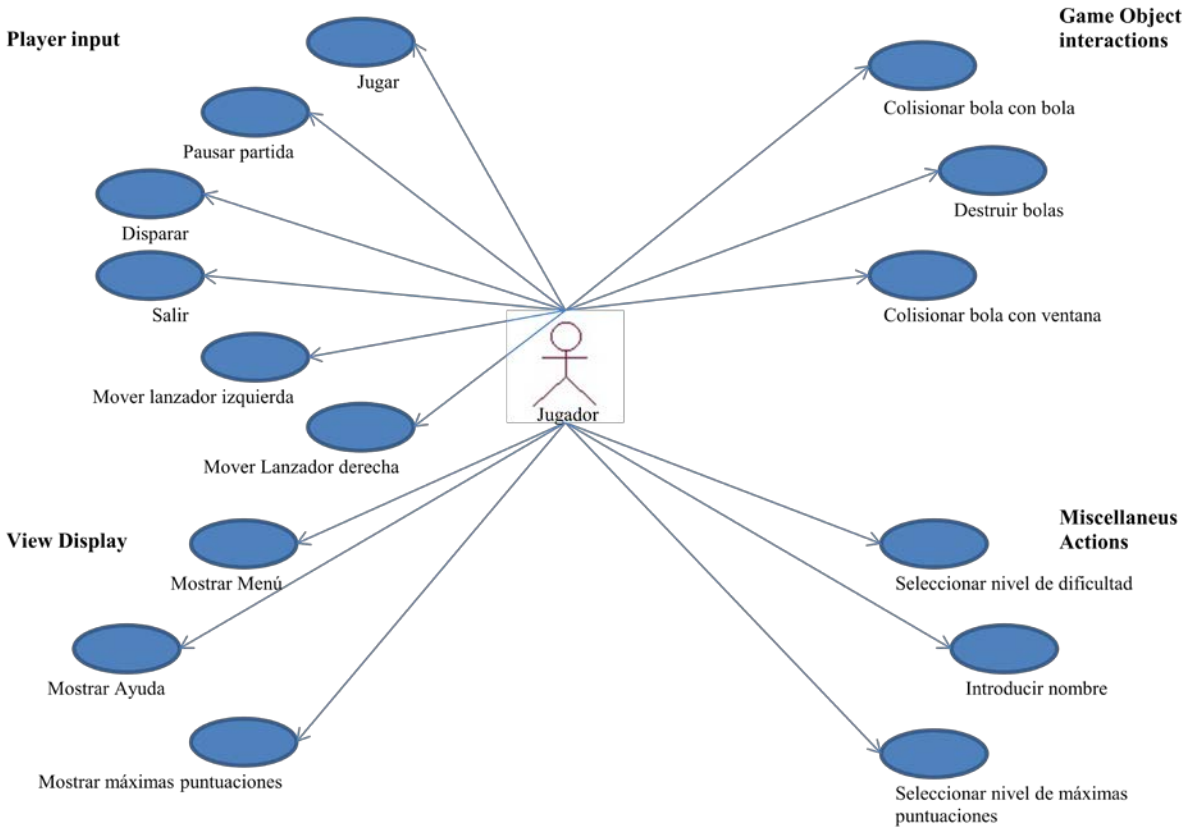

Figura 68. Diagrama de casos de uso

A continuación, se adjuntan las tablas con la especificación textual de cada caso de uso. De cada caso de uso se especificarán los siguientes términos para comprender mejor cada uno de ellos:

- **Identificador**: Representa de forma unívoca cada caso de uso. La sintaxis de nombrado será la siguiente: CU-XX, donde XX serán 2 dígitos que numeran ordenadamente los casos de uso.
- **Nombre**: Título que representa a cada caso de uso, de acuerdo al diagrama anterior.
- **Actores**: Usuarios que intervienen directamente en la realización del caso de uso.
- **Objetivo**: Objetivo concreto del caso de uso.
- **Precondiciones**: Condiciones que deben darse previamente en el sistema para que el caso de uso pueda efectuarse.
- **Postcondiciones**: Indican el estado del sistema después de ejecutarse el caso de uso.

IDENTIFICADOR: CU-01	
Nombre	Jugar
Actores	Jugador
Objetivo	Comenzar una nueva partida de Bubble shooter
Precondiciones	Haber ejecutado el juego
Postcondiciones	Se genera la pantalla inicial y el jugador comienza la partida
IDENTIFICADOR: CU-02	
Nombre	Pausar partida
Actores	Jugador
Objetivo	Parar la partida durante un intervalo de tiempo indeterminado para posteriormente continuarla en el mismo lugar donde se había dejado
Precondiciones	El jugador debe estar jugando una partida
Postcondiciones	La partida continúa donde el jugador la había dejado antes de pausar el juego

IDENTIFICADOR: CU-03	
Nombre	Disparar
Actores	Jugador
Objetivo	El jugador dispara una bola a través del lanzador
Precondiciones	- El jugador ha iniciado una partida - El jugador ha seleccionado el ángulo de disparo
Postcondiciones	La bola disparada se mueve por la ventana hasta colisionar con otra que está fija en el mapa de bolas

IDENTIFICADOR: CU-04	
Nombre	Salir
Actores	Jugador
Objetivo	Terminar el juego
Precondiciones	Haber iniciado el juego
Postcondiciones	Salir del juego

IDENTIFICADOR: CU-05	
Nombre	Mover lanzador a la izquierda
Actores	Jugador
Objetivo	Mover el lanzador a la izquierda para conseguir el ángulo deseado
Precondiciones	El jugador debe estar jugando una partida
Postcondiciones	La flecha del lanzador apunta al ángulo deseado

IDENTIFICADOR: CU-06	
Nombre	Mover lanzador a la derecha
Actores	Jugador
Objetivo	Mover el lanzador a la derecha para conseguir el ángulo deseado
Precondiciones	El jugador debe estar jugando una partida
Postcondiciones	La flecha del lanzador apunta al ángulo deseado

IDENTIFICADOR: CU-07	
Nombre	Mostrar Menú
Actores	Jugador
Objetivo	Mostrar las opciones del juego
Precondiciones	Haber iniciado el juego
Postcondiciones	Se muestran las opciones que tiene el juego

IDENTIFICADOR: CU-08	
Nombre	Mostrar puntuaciones máximas
Actores	Jugador
Objetivo	Mostrar las puntuaciones máximas dentro de un nivel
Precondiciones	Haber seleccionado un nivel
Postcondiciones	Muestra en pantalla las 10 puntuaciones máximas del nivel escogido

IDENTIFICADOR: CU-09	
Nombre	Mostrar Ayuda
Actores	Jugador
Objetivo	Informar al jugador de la mecánica del juego y los controles
Precondiciones	Haber seleccionado la opción de Ayuda en el menú principal
Postcondiciones	Se muestra en pantalla la mecánica del juego y los controles a utilizar

IDENTIFICADOR: CU-10	
Nombre	Colisionar Bola con Bola
Actores	Jugador
Objetivo	Colisionar la bola disparada por el lanzador con otra situada en el mapa
Precondiciones	Haber disparado una bola con el lanzador
Postcondiciones	La bola disparada colisiona con otra del mapa dándose tres posibles situaciones: o La bola disparada se sitúa a la izquierda de la bola con la que colisiona o La bola disparada se sitúa a la derecha de la bola con la que colisiona o La bola disparada se sitúa debajo y a la izquierda de la bola con la que colisiona o La bola disparada se sitúa debajo y a la derecha de la bola con la que colisiona

IDENTIFICADOR: CU-11	
Nombre	Destruir bolas
Actores	Jugador
Objetivo	Destruir todas las bolas que son del mismo color
Precondiciones	Haber colisionado una bola con otra bola
Postcondiciones	La bola disparada al colisionar con otra del mismo color, si el número de bolas agrupadas del mismo color que la que colisiona suman más de tres, se produce la explosión de todas ellas desapareciendo del mapa

IDENTIFICADOR: CU-12	
Nombre	Colisionar Bola con ventana
Actores	Jugador
Objetivo	Rebotar la bola dentro de la ventana al colisionar con ésta
Precondiciones	Haber disparado una bola con el lanzador
Postcondiciones	Cuando una bola colisiona con las paredes laterales de la ventana del juego, produce un rebote de ésta hacia arriba

IDENTIFICADOR: CU-13	
Nombre	Introducir Nombre
Actores	Jugador
Objetivo	Introducir el nombre del jugador en la lista de máximas puntuaciones
Precondiciones	Haber conseguido una puntuación máxima en el nivel jugado
Postcondiciones	Se almacena el nombre del jugador y su puntuación en la lista de máximas puntuaciones del nivel en el que ha jugado

IDENTIFICADOR: CU-14	
Nombre	Seleccionar Nivel de Dificultad
Actores	Jugador
Objetivo	Seleccionar el nivel de dificultad del juego
Precondiciones	Seleccionar la opción de Nivel de Dificultad en el menú principal
Postcondiciones	Cuando se inicie la partida, se hará en el nivel seleccionado

IDENTIFICADOR: CU-15	
Nombre	Seleccionar Nivel de Máximas puntuaciones
Actores	Jugador
Objetivo	Ver la lista de máximas puntuaciones del nivel seleccionado
Precondiciones	Seleccionar la opción de Máximas puntuaciones en el menú principal
Postcondiciones	Se muestra en pantalla la lista de máximas puntuaciones del nivel seleccionado

3.1.4. Diagrama de actividad del sistema

Una vez especificados los casos de uso, los cuales representan la interacción entre usuario y sistema, quedan más claras las distintas funcionalidades que el usuario puede realizar. Para que quede más claro el comportamiento del sistema se mostrará a continuación su diagrama de actividad, que recoge las transiciones y eventos.

Los diagramas de actividad describen la secuencia de las actividades en un sistema. En ellos, las transiciones entre estados se producen como consecuencia de eventos y pueden tener un procesamiento asociado. Representan los sucesos que influyen en el comportamiento y evolución del sistema, describiendo tanto su comportamiento normal (ejecución típica), como excepcional (errores o excepciones). A continuación, en la figura 69, se muestra el diagrama actividad del sistema:

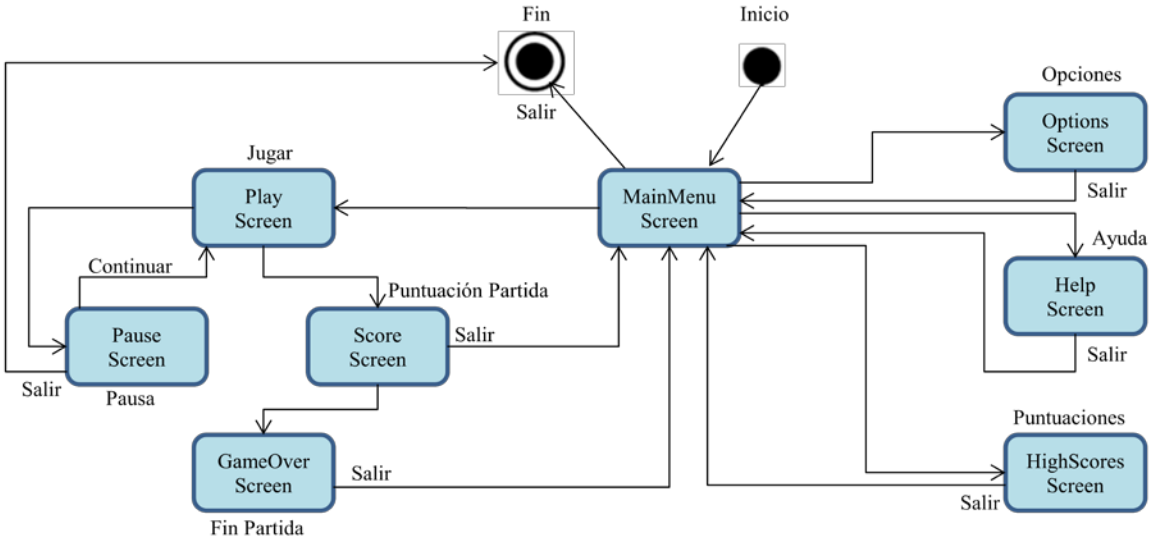

Figura 69. Diagrama de actividad del sistema

3.1.5. Diagramas de secuencia

La especificación de los casos de uso ha permitido dar una idea clara de cómo el usuario puede utilizar el sistema a través de sus distintas funcionalidades. De la misma manera, el diagrama de actividad del sistema ha aclarado cuál es el comportamiento del sistema ante los eventos más típicos.

Los diagramas de secuencia, permiten representar cómo se comunican entre sí varios objetos para la realización de los casos de uso. Si el diagrama de casos de uso permite el modelado de la vista o escenario del sistema, el diagrama de secuencia contiene detalles de

implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos; de manera que se puede determinar qué objetos son necesarios para la implementación del escenario. Los diagramas de secuencia facilitarán la tarea de creación del modelo de clases en la fase de diseño.

Sería muy pesado representar el diagrama de secuencia de cada caso de uso ya que por su similitud muchos no aportarían ninguna información de interés. Por esto, se ha decidido que sólo se van a recoger en este punto algunos diagramas de secuencia que hagan referencia a funcionalidades importantes del sistema.

La figura 70 muestra el diagrama de secuencia de la funcionalidad de **Jugar**, en la que el jugador lanza el juego y comienza una nueva partida. Se puede observar la interacción de la clase *ScreenManager*, que actúa como gestor de pantallas de la aplicación, cargando el Menú Principal. Al seleccionar Jugar desde dicho menú, lanza un *GameScreen* o pantalla de juego genérica. Esta mediante el método LoadContent lanza la pantalla de juego de *Bubble Shooter*, que tras cargar las texturas inicializa un nuevo jugador y las bolas que se cargan en pantalla. Durante el juego, el jugador dispara bolas, y éstas tienen las funciones de posicionarse en el mapa de bolas o explotar.

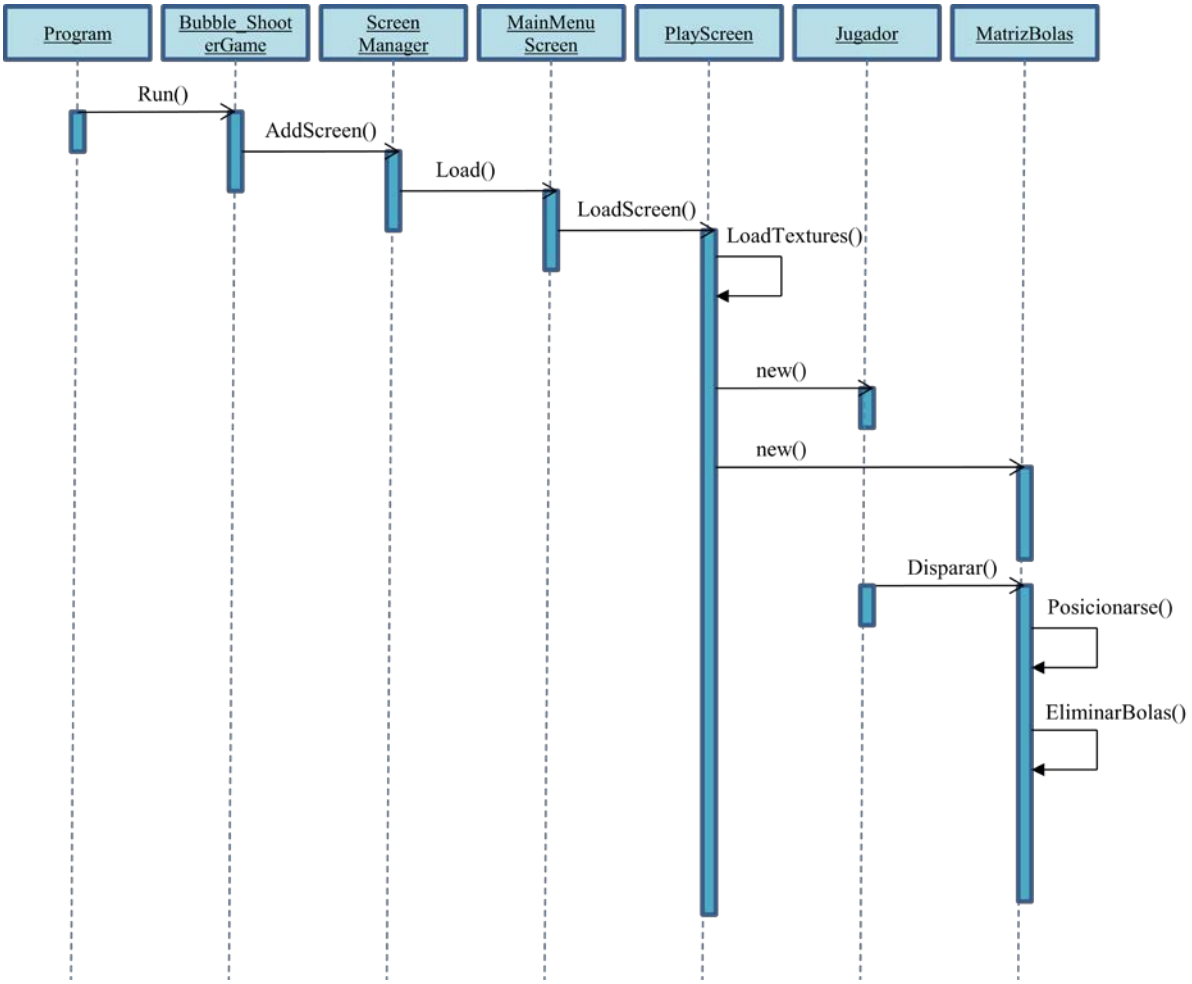

Figura 70. Diagrama de secuencia de Jugar

En el diagrama de secuencia de la figura 71 se representa la funcionalidad de dibujar un *Frame*[35], que actualiza el frame de la animación después de un movimiento. La secuencia de llamadas es muy simple, tras realizar un movimiento la clase *PlayScreen* llama al método Update de Jugador. Tanto el jugador como las bolas heredan de *Sprite* que es la clase que ejecuta el método *Draw*; método en el que se actualiza al siguiente frame de la animación correspondiente para que posteriormente se muestre por pantalla cuando la clase *Game* ejecute su propio método Draw.

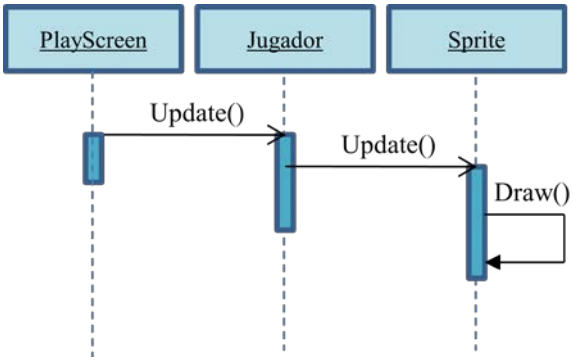

Figura 71. Diagrama de secuencia de Dibujar

La figura 72 muestra el diagrama de secuencia del disparo de una bola por parte del lanzador. *PlayScreen* detecta que se ha producido un movimiento y hace Update de *Jugador*. La clase *Jugador* ejecuta *DispararCohete*, produciendo el movimiento de la bola por la ventana. Finalmente, se actualizan las texturas del lanzador y la bola base, que es la que muestra el color de la siguiente bola llamando al método de *ModificarSprite*.

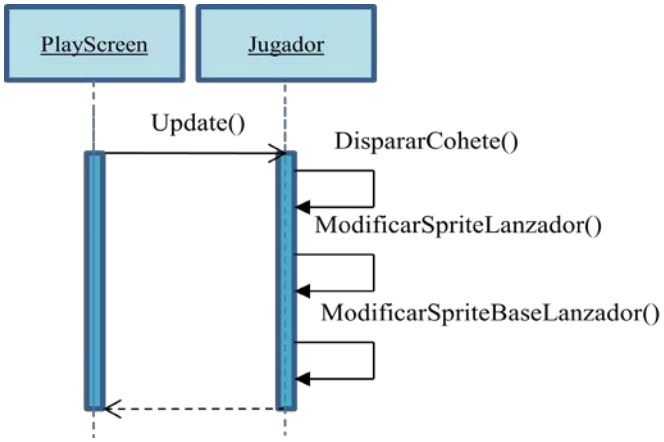

Figura 72. Diagrama de secuencia del disparo de una bola

En la figura 73 se representa el diagrama de secuencia del movimiento de las bolas. *PlayScreen* detecta que se ha producido un movimiento y hace Update de *Jugador*. La clase *Jugador* ejecuta *MoverCohete*, método que comprueba que el movimiento no es el mismo que el anterior para continuar con el siguiente frame de una animación, o en caso contrario lanzar una nueva. Mientras actualiza el frame comprueba que la bola no toque con las paredes de la ventana de juego, en ese caso, continúa con el mismo sentido hacia arriba, pero modifica la dirección de la bola para que continúe dentro de la ventana manteniendo el ángulo. Tras esta serie de llamadas *PlayScreen* ejecutará el método comentado en el anterior diagrama de secuencia (Update), actualizando así la animación correspondiente.

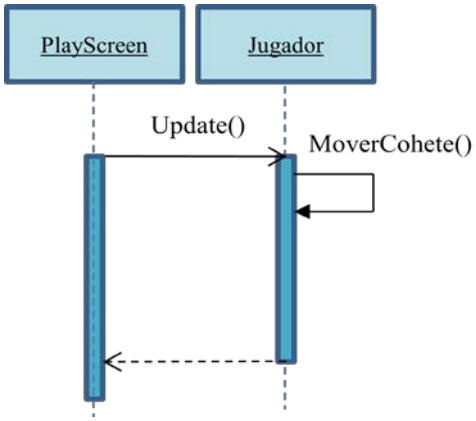

Figura 73. Diagrama de secuencia del movimiento de una bola disparada

La siguiente figura representa lo que ocurre cuando una bola colisiona con otra. En este caso, el método *ActualizarCohetes* realiza la comprobación de las colisiones. Cuando una bola colisiona con otra dentro del mapa, se realiza la comprobación para ver qué lugar debe ocupar esta bola en el mapa, lo cual lo realiza este mismo método. Una vez localizada la posición, se añade la bola físicamente al mapa y lógicamente a la matriz donde están representadas las bolas del mapa. La bola una vez posicionada comienza el proceso de búsqueda de bolas contiguas a la nueva bola posicionada en el mapa que sean del mismo color y estén agrupadas. Si el número de bolas encontradas es mayor de 3, se llama al método de *EliminarBolas*, que se encarga de eliminar las bolas seleccionadas y actualizar el nuevo mapa.

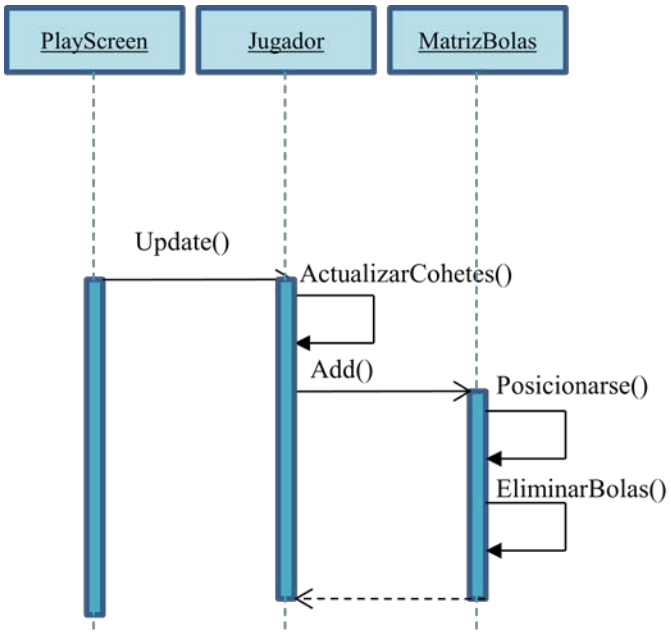

Figura 74. Diagrama de secuencia del proceso de eliminación de bolas

3.2. Fase de Diseño

En la fase de diseño se debe tomar el análisis de la etapa anterior y realizar un diseño exhaustivo de los componentes, para que después en la etapa posterior de implementación esté todo tan claro como para no tener que replantear ningún aspecto. Por tanto en esta fase, hay que centrarse en obtener un diseño que se adapte a las necesidades de la aplicación y una definición exhaustiva de las clases, así como que su respectivo diagrama sirva para ver la colaboración e interacción de éstas.

La primera tarea a llevar a cabo es definir las clases a utilizar y establecer las relaciones entre éstas, que reflejen la estructura a seguir en la Fase de Implementación.

3.2.1. Diagrama de clases

El diagrama de clases describe las clases y objetos que debe tener el sistema y las diversas relaciones de carácter estático que existen entre estas. Es necesario además, para mayor claridad, señalar los atributos y operaciones más importantes de las distintas clases; así como las restricciones de visibilidad a la que se verán sujetas.

Una vez estudiado los requisitos funcionales del sistema, las funcionalidades y el comportamiento del sistema, mediante el análisis en el punto anterior, se está en disposición de comenzar a definir las diferentes clases que compondrán el modelo de diseño, de forma que cubran las necesidades y funcionalidades del software.

En esta sección del capítulo se muestra en primer lugar, el diagrama de clases, la descripción de las clases más importantes y su relación con otras clases y objetos a través de diagramas para posteriormente ir profundizando en las mismas, incluyendo los métodos y propiedades más importantes. La figura 75 muestra el diagrama de clases del proyecto.

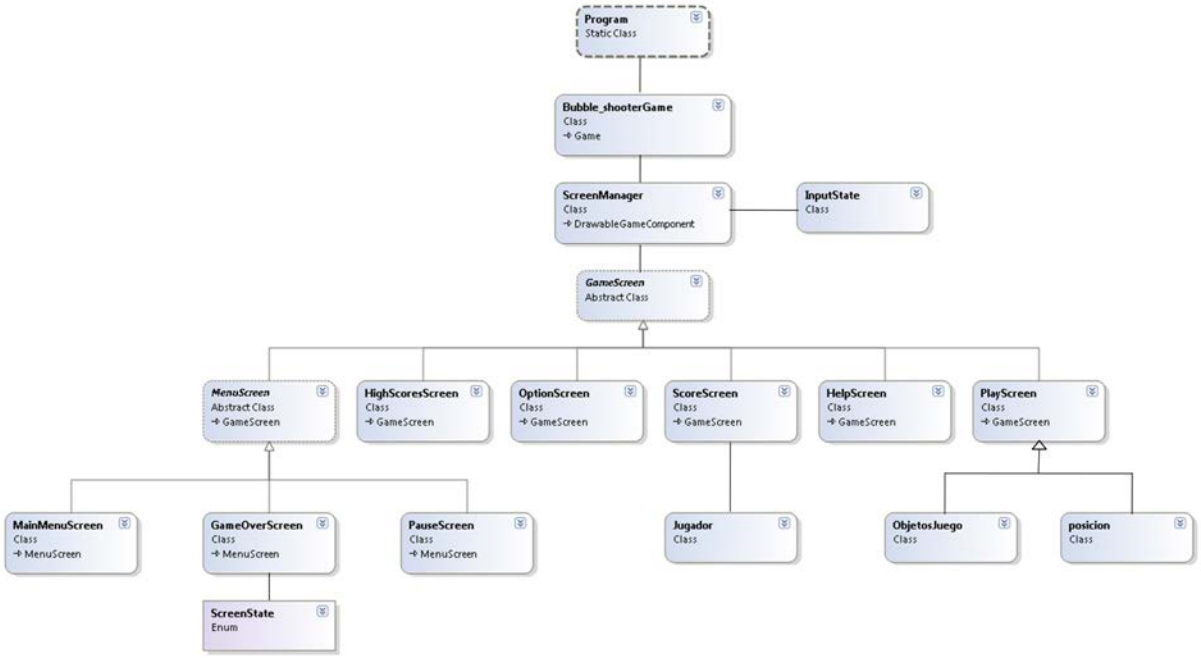

Figura 75. Diagrama de clases

3.2.2. Definición de las clases

Como se puede observar, el proyecto está compuesto por 17 clases. A continuación, se explicarán las clases más relevantes con detalle, haciendo referencia a sus atributos y métodos más importantes. Se hará referencia a todas las clases, pero no se describirán en profundidad algunas por dos motivos, el primero es que existen clases redundantes en lo que al diseño se refiere (como por ejemplo las clases que representan cada tipo de pantalla, que son en esencia iguales) y explicando una quedan representadas las demás. El segundo, que durante el desarrollo del proyecto se han utilizado librerías de apoyo de *Microsoft* (como el gestor de pantallas *ScreenManager*), las cuales han sido modificadas para adecuarlas a la funcionalidad del juego. En el caso de estas últimas se explicará su funcionalidad y los métodos añadidos o modificados con respecto a la librería original.

- **Program**

  Contiene el Main del proyecto, instancia el juego (clase **Bubble_shooterGame**) y llama a su método Run().

- **Bubble_shooterGame**

  Clase que hereda de *Microsoft.Xna.Framework.Game* y proporciona una serie de métodos para inicializar el juego, así como el bucle principal que lo controla. Los atributos más importantes son:

  - *graphics*: Este objeto, de tipo *GraphicsDeviceManager*, es el handler o manejador para la capa de gráficos.

- *spriteBatch*: Objeto de tipo *SpriteBatch* usado para dibujar texto e imágenes en 2D.

En cuanto a los métodos más importantes:

- *Constructor*: Método en el que se asigna la carpeta Content, directorio raíz del proyecto donde se guardan todos los contenidos del juego, es decir, sonidos, gráficos, modelos 3D.
- *Initialize*: dentro de este método se inicializa cualquier cosa que no sea recursos gráficos.
- *LoadContent*: es el lugar indicado para cargar los recursos gráficos.
- *UnloadContent*: en este método se liberan los recursos cargados.
- *Update*: es el bucle del juego, sirve para actualizar los cálculos de los objetos en función de las posiciones y de los inputs del usuario, redibujar la pantalla y tiene una condición de salida que finaliza el juego (el usuario pulsa salir, ha muerto o ha ganado).
- *Draw*: en este método es donde se pintan los sprites a nivel de render de objetos por pantalla.

- **ScreenManager, InputState y GameScreen**

Para el correcto funcionamiento del juego, es necesario un componente que administre la transición entre pantallas, así como la actualización y el dibujado de cada pantalla individual. Para ello se ha tomado como base una implementación propuesta por *Microsoft*, llamada Game State Management. Dicha implementación proporciona un sistema de pantallas que facilita la transición entre cada una de ellas, generando su lógica y contenido de manera independiente. La clase *ScreenManager* esta implementada como un componente dentro del paradigma de *XNA*. Su función es administrar la presentación, superposición y actualización de las pantallas, así como determinar cuál de ellas es la pantalla activa. Además incluye una instancia de la clase *InputState* la cual permite hacer una abstracción de toda señal de entrada, ya sea teclado, mando de *XBOX 360* o *Zune*; pudiendo ser usada por cada pantalla que herede de la clase GameScreen, con la que se proporcionará la funcionalidad básica de inicialización, actualización, manejo de datos de entrada y dibujo.

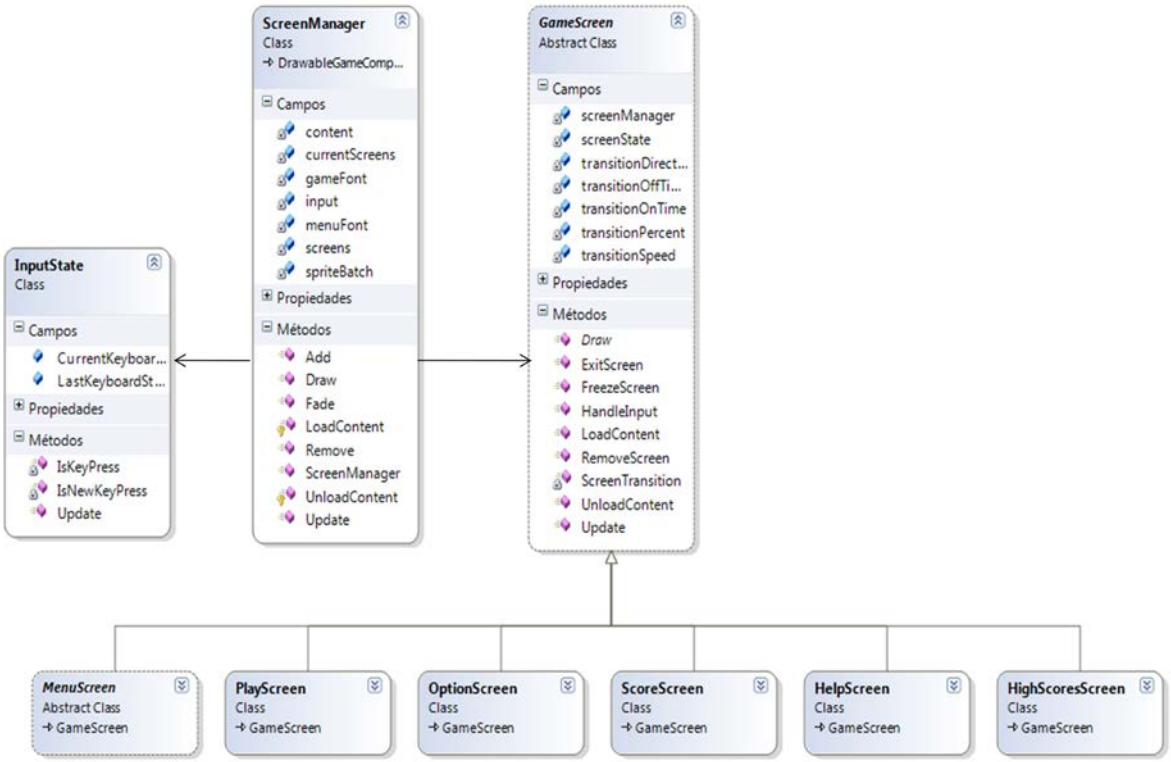

Figura 76. Detalle de las clases del paquete ScreenManager

- **Screens**

  Junto con el paquete de clases *ScreenManager*, el componente Game State Management proporciona un paquete adicional denominado Screens, con modelos básicos de pantallas que heredan de la clase *GameScreen*. Cada una de las clases del paquete Screens representa un tipo de pantalla con funcionalidades muy concretas. Los distintos modelos de pantallas son los siguientes:

  - *MenuScreen*: Es la clase principal del menú en la que se pueden cargar fondos de pantalla, manejar las entradas y manejar los eventos que activan cada una de éstas. En este juego la clase MenuScreen extiende a tres clases que cargan pantallas.
  - *MainMenuScreen*: Esta clase muestra la pantalla del menú inicial extendida de MenuScreen, en la que se muestran las opciones por las que el jugador puede navegar.
  - *GameOverScreen*: Clase extendida de MenuScreen cuya pantalla se muestra cuando finaliza la partida y le da al jugador las opciones de volver al menú principal o salir del juego.
  - *PauseScreen*: Última de las clases extendidas de MenuScreen. Esta clase congela la imagen de la partida en el momento en el que el

jugador pulsa ESCAPE. La partida se queda en estado de espera y permite al jugador continuar con la partida o salir del juego.

- *PlayScreen*: Es la clase que contiene toda la lógica del juego en sí. Carga todas las texturas de los objetos del juego, los efectos de sonido y la matriz del juego como elementos más importantes.
- *OptionScreen*: Clase que carga el nivel de dificultad en la que el jugador va a jugar una partida. El parámetro se le pasa a la clase MainMenuScreen que posteriormente gestionará la opción elegida.
- *ScoreScreen*: Esta clase se instancia al finalizar una partida, y contiene los métodos necesarios para que el jugador introduzca su nombre en la tabla de máximas puntuaciones en caso de conseguir una puntuación lo suficientemente alta. Los datos se almacenan en un fichero, el cual se carga en memoria, se recorre en busca de una posición donde insertar la nueva puntuación; en caso de encontrarla, el jugador podrá introducir su nombre.
- *HighScoresScreen*: Es la clase que se encarga de mostrar las puntuaciones máximas. El jugador a través del teclado escoge el nivel de las puntuaciones máximas que quiere consultar, parámetro que sirve para cargar el fichero que contiene la información. Se recorre y se muestran los datos por pantalla.
- *HelpScreen*: Esta es la clase más sencilla de todo el juego, ya que simplemente su función es la de cargar en pantalla la descripción del juego y los controles.

A grandes rasgos se ha hecho una descripción de las clases más importantes de las que consta el juego, sin embargo, existen otras clases menores que sirven de apoyo a las que se acaban de nombrar. A continuación, y por destacar su importancia, se va a detallar más concretamente las funciones de las dos clases más importantes del juego que son **MenuScreen** y **PlayScreen**.

- **MenuScreen**

  Como anteriormente se ha dicho, la clase MenuScreen es la clase principal sobre la que se apoyan otras tres clases que cargan pantallas en el juego y a través de esta clase el jugador se puede mover a través de las opciones del menú y seleccionar la opción que desee. La siguiente figura muestra la definición de la clase expandida:

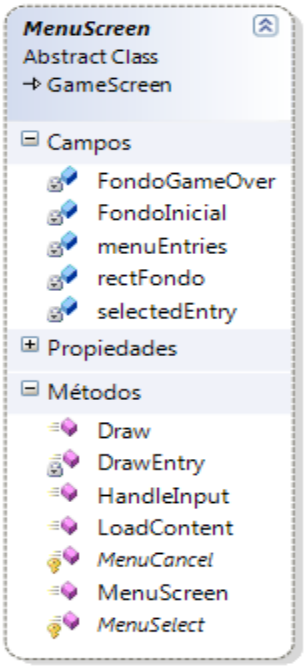

Figura 77. Definición de la clase MenuScreen

Los atributos más importantes son los siguientes:

- menuEntries: Este atributo genera la lista de las opciones que contendrá el menú y que el jugador verá al principio del juego, y en los submenús que seleccione y que tengan una lista de opciones a elegir.
- selectedEntry: Atributo de tipo entero que hace referencia a la entrada del menú que en cada momento está activa.

En cuanto a los métodos hay que destacar los siguientes:

- Constructor: Inicializa la pantalla y asigna un tiempo mínimo de transición en la interacción entre pantallas.
- HandleInput: Se encarga de manejar las entradas del jugador y marca la opción elegida del menú.
- LoadContent: Carga las texturas y el rectángulo en el que se contendrán éstas de las pantallas inicial y de fin de juego.
- Draw: Método donde se pinta en pantalla todo lo que se ha generado en el código.

- **PlayScreen**

  Esta es la clase más importante de todo el proyecto, ya que contiene toda la lógica del juego y de ahí que sea la clase más extensa de todas. Esta clase hereda de la clase GameScreen y se apoya en las clases *ObjetosJuego* y p*osicion* para confeccionar los métodos que contiene PlayScreen.

  Una vez que se selecciona la opción "*Jugar*" del menú principal se instancia P*layScreen*, y se cargan las texturas, las animaciones, los efectos sonoros, se genera el nivel correspondiente y se inicializan todos los elementos del juego y del jugador. La mayoría de estos elementos, como las texturas de jugador o los efectos sonoros son comunes a todos los niveles por lo que no se vuelven a cargar.

  Debido a la complejidad de la clase en cuanto al número de atributos y métodos, se destacarán los más importantes por funcionalidad para ofrecer una visión más clara y concisa de lo que se ha querido realizar. La siguiente figura muestra la definición de la clase expandida sólo en los métodos debido a su gran extensión:

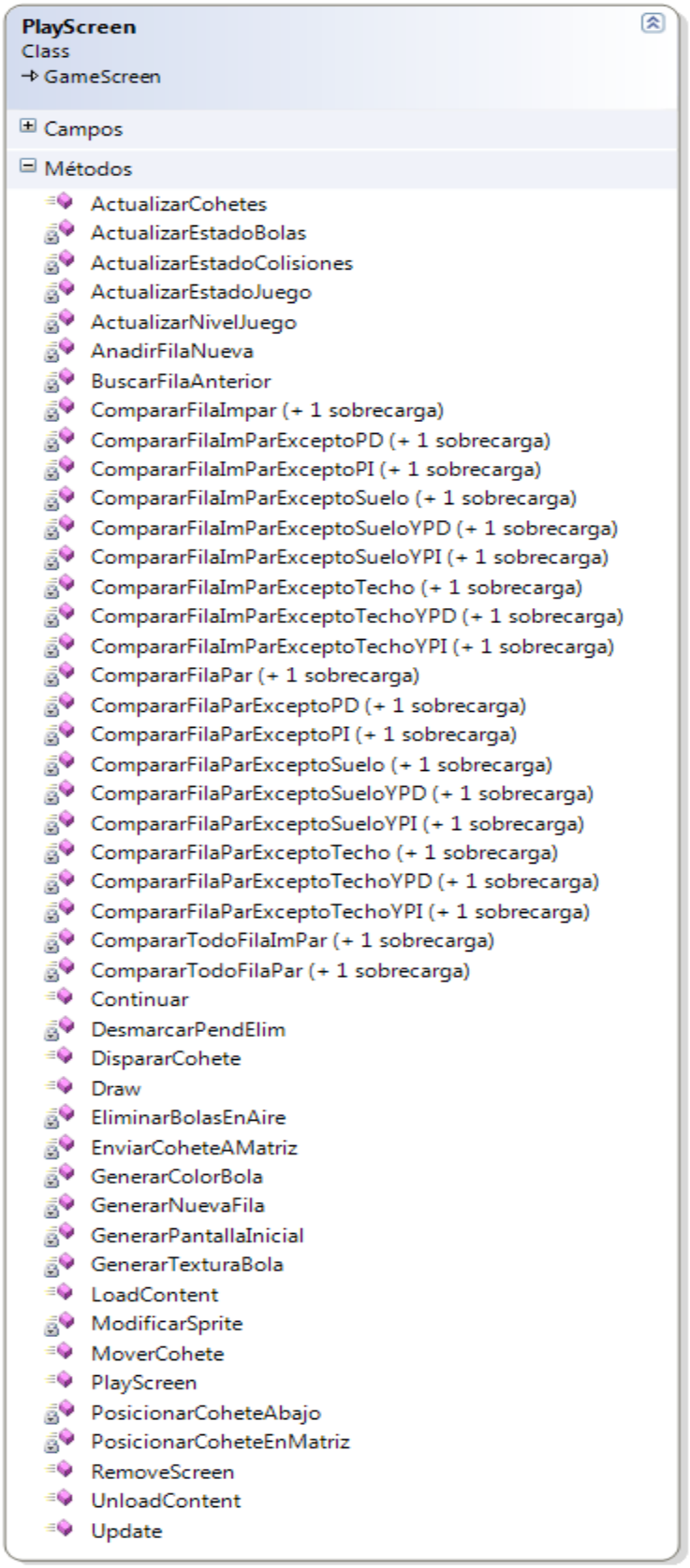

Figura 78. Definición de la clase PlayScreen

Los atributos más importantes son por tanto los siguientes:

- Texturas: Son todos los atributos de tipo *Texture2D* que cargan los sprites que forman parte del juego como son las bolas, el lanzador, la base del lanzador, el cohete, el fondo, etc.
- ObjetosJuego: Son los elementos más importantes ya que los elementos que son de este tipo contienen a su vez los atributos con los que se gestionan las decisiones del juego. La clase ObjetosJuego se explicará más adelante.
- ArrayList: Las lista que contiene esta clase almacena entre disparo y disparo las bolas de la matriz susceptibles de ser eliminadas cuando el cohete colisiona con una bola de la matriz.
- SoundEffect: Los objetos del tipo de efectos de sonido, cargan los sonidos añadidos en el proyecto. Los efectos sonoros que contiene el juego se producen en los siguientes casos:
  - Cuando se dispara una bola (sale el cohete del lanzador).
  - Cuando se posiciona el cohete en la matriz.
  - Cuando se elimina una bola de la matriz.
  - Cuando la matriz queda limpia de bolas.
- posicion: Los objetos de tipo posición contienen una posición bidimensional que sirve para saber en cada momento la posición que ocupan las bolas en la matriz.
- Vectores: Los vectores son de tipo *Vector2D* y se encargan de gestionar la posición de los objetos dentro de la ventana del juego.
- BoundingBox y Rectangle. Son los objetos que gestionan la detección de colisiones.

Como se puede apreciar, la clase está constituida por un gran número de métodos, los cuales para no extender demasiado el punto se tratarán los más importantes:

- Constructor: Recibe como parámetro el nivel de juego seleccionado por el jugador y se selecciona el número de segundos que transcurren hasta que se genera otro nivel de bolas.
- ActualizarCohetes: Método llamado desde Update. Actualiza el movimiento del cohete con el ángulo indicado y lo envía a la matriz cuando detecta una colisión con una bola de la matriz.

- ActualizarEstadoBolas: Función recursiva que recibe 2 listas dinámicas y el color del cohete que ha sido posicionado. A través del algoritmo de búsqueda almacenará en las listas los objetos correspondientes a las bolas adyacentes del mismo color que el cohete.
- ActualizarEstadoColisiones: El método recibe la posición del cohete posicionado en una celda y busca a través de un algoritmo recursivo todas las bolas adyacentes del mismo color a la del cohete posicionado. Si encuentra 3 ó mas las elimina de la matriz.
- BuscarFilaAnterior: Algoritmo que busca la fila anterior de una bola situada en una zona determinada del array.
- CompararFilas: Los métodos de CompararFilas se resumen en este punto para no hablar de los 40 métodos de los que consta. Todos los métodos involucrados de este tipo forman parte del algoritmo diseñado para la explosión de las bolas. El algoritmo se explicará con detalle en otro punto de la memoria.
- DispararCohete: Método llamado desde *Update* para lanzar una bola (cohete) con la dirección y el ángulo que indica el lanzador.
- EliminarBolasEnAire: Método llamado desde el algoritmo de explosión de bolas el cuál elimina todas las bolas marcadas en una lista de array que ha ido añadiendo las bolas o grupo de bolas que quedan en el aire, es decir, que quedan descolgadas.
- EnviarCoheteAMatriz: Método llamado cuando el cohete colisiona con un objeto de la matriz. Recibe la posición de la celda del objeto colisionado y posiciona el cohete en el lugar donde le corresponda dentro de la matriz.
- GenerarNuevaFila: La función genera una nueva fila de bolas en la matriz cuando se cumple el tiempo determinado por el nivel.
- GenerarPantallaInicial: Genera la matriz inicial de *ObjetosJuego* con sus propiedades para comenzar la partida.
- LoadContent: Se ejecuta al comenzar la partida y su cometido es cargar el fondo de la pantalla, las texturas de todos los objetos del juego y los efectos de sonido; genera la pantalla inicial; y por último carga e inicializa al lanzador y la base del lanzador.
- ModificarSprite: Método que carga la textura de un objeto con el color que se le ha pasado como parámetro. En caso de que el parámetro "*lanzador*" sea true, la textura que se carga corresponde a la del lanzador, si no, corresponde a una bola normal.

- MoverCohete: Realiza el movimiento del cohete y controla que no se salga de pantalla, haciendo que rebote en el marco de la ventana del juego siempre hacia arriba y con el ángulo en el que viaja.
- PosicionarCoheteEnMatriz: Método que recibe las posiciones donde será incluido el cohete en la matriz. Se añaden las propiedades al objeto de la celda y se desactiva el cohete.
- Update: Controla el *tempo* del juego y los movimientos del jugador en cada ciclo de reloj.

Como se ha dicho anteriormente, dentro de la clase PlayScreen existen otras clases fundamentales sobre las que se apoya, pero por importancia sólo se describirá la clase **ObjetosJuego.**

- **ObjetosJuego**

  La clase *ObjetosJuego* es la clase que contiene todos los datos necesarios de los componentes del juego (lanzador, base lanzador, cohete y matriz de objetos) para gestionar la partida. La figura 79 muestra la definición de la clase.

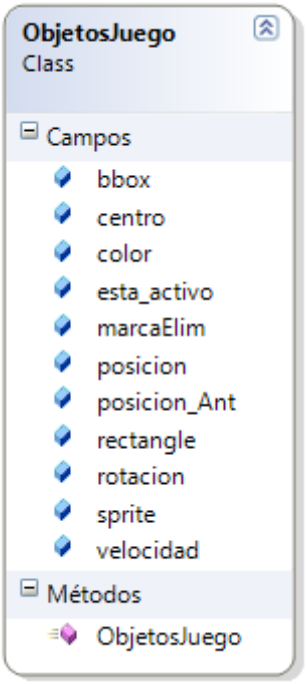

Figura 79. Definición de la clase ObjetosJuego

Los atributos de los que consta la clase son los siguientes:

- sprite: Atributo de tipo Texture2D que contendrá la textura del objeto.
- posicion: Indica la posición en la que se encuentra el objeto dentro de la pantalla de juego.
- Posicion_Ant: Lugar en el que se encontraba el objeto en el ciclo de reloj anterior.
- centro: Centro de rotación del objeto. Sólo aplicable al lanzador.
- rotacion: Indica el ángulo de rotación del lanzador y del cohete.
- Esta_activo: Atributo booleano que indica si el objeto está activo o no.
- velocidad: Dirección y velocidad a la que viaja el cohete.
- Rectangle y bbox: Atributos utilizados para la detección de colisiones.
- color: Color de la bola.
- marcaElim: Atributo booleano que indica si el objeto es susceptible de ser eliminado.

Una vez descritos los atributos, la clase sólo contiene un único método constructor que inicializa todos los parámetros anteriores.

3.3. Fase de Implementación

En este punto se mostrará cómo se va a utilizar el análisis y diseño llevados a cabo en los apartados 3.1 y 3.2 respectivamente, implementando las clases propuestas y teniendo en cuenta que la aplicación refleje las funciones presentadas en los Casos de Uso. Por otro lado, es necesario destacar que la finalidad del punto no es la de comentar el código de las clases, si no profundizar en todos los procesos que componen la fase de desarrollo. Por tanto, se especificará que sistema se ha utilizado para la detección de colisiones de las bolas; cómo se ha diseñado e implementado el mapa de bolas y su comportamiento durante el juego; se detallarán los sprites que se han usado para las animaciones del jugador y las bolas, se especificará el diseño de los niveles y se explicará cómo el diseño permitiría añadir nuevas funcionalidades de forma sencilla. En definitiva cualquier aspecto relacionado con el desarrollo que se considere importante.

En cuanto a la organización del capítulo, será paralela al desarrollo real del juego, de forma que se comenzará por comentar aspectos de la implementación del jugador, posteriormente del entorno y los niveles, después de las bolas y por último de otros elementos adicionales de relevancia.

3.3.1. Elementos que intervienen en el juego

Para la implementación del juego se tomó como punto de partida el jugador, buscando tres requisitos fundamentales para que fuese lo más dinámico posible:

- Conseguir un desplazamiento fluido del jugador en pantalla, sin cortes ni ralentizaciones.
- Animar los movimientos de forma correcta.
- Una vez implementados los movimientos, marcar sus límites y detectar las colisiones de las bolas lanzadas por el jugador con las demás bolas del mapa.

A continuación se especifican las características más importantes del jugador, para comprender mejor cómo realiza los movimientos.

- Sprites

La funcionalidad completa del jugador (lanzador) se consigue gracias a las clases *posicion* y *ObjetosJuego*. El primer paso es mostrar al jugador en la pantalla y se realiza a través de los métodos de la clase ObjetosJuego. Para conseguir mostrar un elemento en pantalla es necesario cargar la imagen correspondiente usando el método Load en un objeto de tipo **Texture2D**.

La gestión de contenido gráfico y sonoro de un proyecto en *XNA*, se realiza a través del *Content Pipeline (API* que permite a los desarrolladores y diseñadores incorporar contenidos multimedia en los proyectos creados con *XNA* framework explicados en el capítulo 2). El *Content Pipeline* o CPL proporciona el importador (es el encargado de obtener los datos y normalizarlos) y el procesador (recibe los datos que el importador ha generado y crea un objeto para su uso en tiempo de ejecución). En este caso el importador será el objeto de tipo *ContentManager* y los procesadores dependen de los datos que se quieran cargar.

El jugador en este juego, debido a su funcionalidad puede contener 6 texturas distintas, ya que cuando dispara una bola, la textura siguiente que contendrá será la que le indique el objeto *baselanzador* antes de haber lanzado la bola. La carga de texturas tanto de las bolas como de los lanzadores se muestra en la siguiente tabla de código:

```
ContentManager content;
content = new ContentManager(game.Services, "Content");

Texture2D textAmarilla = ScreenManager.Content.Load<Texture2D>("Bolas/amarilla");
Texture2D textLanzAmarilla =
ScreenManager.Content.Load<Texture2D>("Bolas/amarilla1");
Texture2D textAzul = ScreenManager.Content.Load<Texture2D>("Bolas/azul");
Texture2D textLanzAzul = ScreenManager.Content.Load<Texture2D>("Bolas/azul1");
Texture2D textCeleste = ScreenManager.Content.Load<Texture2D>("Bolas/celeste");
Texture2D textLanzCeleste =
ScreenManager.Content.Load<Texture2D>("Bolas/celeste1");
Texture2D textRoja = ScreenManager.Content.Load<Texture2D>("Bolas/roja");
Texture2D textLanzRoja = ScreenManager.Content.Load<Texture2D>("Bolas/roja1");
Texture2D textVerde = ScreenManager.Content.Load<Texture2D>("Bolas/verde");
Texture2D textLanzVerde = ScreenManager.Content.Load<Texture2D>("Bolas/verde1");
Texture2D textMorada = ScreenManager.Content.Load<Texture2D>("Bolas/morada");
Texture2D textLanzMorada = ScreenManager.Content.Load<Texture2D>("Bolas/morada1");
```

Código 1. Texturas de los elementos del juego

Se puede observar que existen doce texturas diferentes pero sólo seis colores. Las texturas cuyos nombres vienen definidos por text<*Color*> representan las texturas de las bolas, y las texturas cuyos nombres vienen definidos por textLanz<*Color*> representan las texturas que puede adoptar el jugador.

Las figuras 80 y 81 muestran las texturas de los sprites definidas para el lanzador y las bolas respectivamente.

Figura 80. Texturas del lanzador

Figura 81. Texturas de las bolas

Las animaciones del lanzador se basan en moverlo de izquierda a derecha, por lo que no es necesario generar sprites distintos para cada movimiento; para hacerlo más sencillo se ha optado por rotar el sprite a izquierda o derecha según se indique por teclado o ratón.

Lo primero que hay que tener en cuenta es que durante la rotación, la base del sprite no debe girar, por lo que hay que establecer el centro de rotación en el centro de la bola; para poder gestionar

esta situación, la clase que genera los sprites contiene una variable de tipo Vector2 donde se le indica cual es el centro de rotación del sprite, como se muestra en el código 2.

```
centro = new Vector2(sprite.Width / 2, sprite.Height - 15);
```

Código 2. Centro de rotación del lanzador

El siguiente factor es la rotación en sí misma. La variable de rotación se inicializa a 0,0 radianes, que indica la posición vertical (90º) de inicio del lanzador cuando se empieza una nueva partida. En el momento en el que el jugador mueve el cursor o el ratón a izquierda o derecha, por cada pulsación gira el ángulo de rotación 0,1 radianes en negativo si es a la izquierda o en positivo si es a la derecha, limitando el giro entre 30º y 150º.

Además de indicar el ángulo de giro, esta variable indica a la bola (cohete) en el momento del disparo el ángulo de giro que debe tomar. El código 3 muestra esta situación cuando se presionan las flechas izquierda o derecha del teclado.

```
if (estadoTeclado.IsKeyDown(Keys.Left)) lanzador.rotacion -= 0.1f;
//Gira el lanzador hacia la izquierda
if (estadoTeclado.IsKeyDown(Keys.Right)) lanzador.rotacion += 0.1f;
//Gira el lanzador hacia la derecha
```

Código 3. Rotación del lanzador

➢ Disparo

En el momento en el que se produce el disparo, se crea un nuevo objeto (cohete) de tipo *ObjetosJuego* que parte de la posición base del lanzador. Al nuevo cohete se le asigna la propiedad *esta_activo* a **true** para que no sea posible generar otro cohete mientras éste se esté desplazando por la ventana. La textura se genera mediante el método *GenerarTexturaBola* que toma como parámetro el color de la textura del lanzador.

A continuación, al parámetro de velocidad se le asigna la fórmula para generar el ángulo en el que tiene que desplazarse y la velocidad a la que tiene que moverse. Como puede observarse en el código 4, para generar el ángulo, las funciones matemáticas de Seno y Coseno toman el valor del parámetro de rotación del lanzador.

Para que el cohete salga con el mismo ángulo de rotación que la base del lanzador y produzca el efecto de que realmente es la bola del lanzador la que sale despedida, al cohete se le asigna el mismo ángulo de rotación que tiene el lanzador, aunque en el momento en el que la bolsa colisiona con otra en el mapa y se posiciona en éste, adopta la rotación de las demás.

```
//Cargar e inicializar los cohetes
Texture2D texturaCohete = GenerarTexturaBola(colorActual);
cohete = new ObjetosJuego(texturaCohete, colorActual);

cohete.esta_activo = true; //Activa el cohete (para que sea dibujado)
ModificarSprite(cohete, lanzador.color, false); //Cambiamos el color
del sprite para que sea el mismo que el lanzador
cohete.posicion = lanzador.posicion; //El cohete inicia en la posición
del lanzador
cohete.velocidad = new Vector2((float)Math.Sin(lanzador.rotacion), -
(float)Math.Cos(lanzador.rotacion)) * 12.0f; //En qué dirección y a qué
velocidad sale el cohete. Hay que tener en cuenta la rotación del
lanzador
cohete.rotacion = lanzador.rotacion; //Rotamos el sprite del cohete
para que coincida con el del lanzador

//Añadimos los parámetros referentes a la detección de colisiones
cohete.rectangle = new Rectangle((int)cohete.posicion.X,
(int)cohete.posicion.Y, cohete.sprite.Width, cohete.sprite.Height);
cohete.bbox = new BoundingBox(new Vector3(cohete.posicion, 0), new
Vector3(cohete.posicion.X + cohete.sprite.Width, cohete.posicion.Y +
cohete.sprite.Height, 0));

//Actualizamos la textura del lanzador y baselanzador
colorActual = colorSig;
colorSig = GenerarColorBola();
ModificarSprite(lanzador, baselanzador.color, true);
ModificarSprite(baselanzador, colorSig, false);
```

Código 4. Función del momento de disparo de un cohete

➢ Movimientos

Una vez cargadas las animaciones se deben implementar los movimientos. Para realizar correctamente el movimiento hay que controlar dos factores, el primero la entrada de usuario y el segundo el desplazamiento del jugador en respuesta a esa entrada.

En el caso de la entrada, como se ha visto anteriormente se trata de controlar el ángulo de giro del lanzador. La posición que va obteniendo el lanzador la va marcando el ángulo de rotación, es decir, el atributo *rotacion* del objeto lanzador, y el atributo *posicion*, que será siempre la misma, ya que tanto el lanzador como la base del lanzador siempre están fijas.

La respuesta a la entrada supone la posición que toma el objeto en un momento determinado. En este juego se producen dos tipos de movimientos, el del lanzador y el del cohete. El del lanzador ya se ha visto anteriormente cómo se realiza, el del cohete se realiza actualizando el atributo *velocidad* del cohete, ya que el vector se actualiza en virtud de esta magnitud. El método empleado para el movimiento de los cohetes es *ActualizarCohetes*, descrito anteriormente, sin embargo, para mejor comprensión, se adjunta el código del método señalado:

```
bool romperBucle = false;
cohete.posicion += cohete.velocidad;
MoverCohete(); //Actualizamos la posición del cohete durante su
movimiento hasta que se posiciona.
//Si el cohete se sale de la pantalla entonces rebota en las paredes
if (cohete.posicion.Y <= 0) //Toca el techo
{
   int posColumna = (int)(cohete.posicion.X) / 30;
   EnviarCoheteAMatriz(-1, posColumna); //Posicionamos el cohete en la
matriz
}
else
{
   for (int f = 0; f < FilasGeneradas; f++)
   {
       for (int c = 0; c < MAXCOLUMNAS; c++)
       {
          //El cohete colisiona con un objeto de la matriz
          if (MatrizBolas[f, c].bbox.Intersects(cohete.bbox) &&
MatrizBolas[f, c].esta_activo)
          {
             //Posicionamos el cohete en la matriz
             EnviarCoheteAMatriz(f, c);
             romperBucle = true;
             break;
          }
       }
       if (romperBucle) break;
    }
}
```

Código 5. Movimiento de un cohete

En este código, se puede observar cómo la posición se va incrementando a sí misma con la velocidad, dando lugar a un movimiento recto en el ángulo de disparo. En caso de que el cohete toque la parte más alta de la ventana del juego (techo) se posiciona, y si colisiona con una bola de la matriz, el método *EnviarCoheteAMatriz* gestiona la colisión.

Durante el movimiento, se va viendo en la pantalla dicho movimiento. Este factor corre a cargo del método *Draw* de la clase *SpriteBatch* que pinta los sprites en la pantalla. Los parámetros que necesita el método para pintar en la pantalla el objeto se describen en el código siguiente poniendo como ejemplo el lanzador:

```
/* Dibuja el lanzador como tal, estos son los parámetros
 * Parámetro 1: El sprite del lanzador
 * Parámetro 2: La posición del lanzador
 * Parámetro 3: null porque es solo una imagen (no una sucesión de
estas
 * Parámetro 4: Color.White, combinación de color
 * Parámetro 5: Angulo de rotación por defecto
 * Parámetro 6: Centro, el centro del sprite para girarlo después
 * Parámetro 7: Escala a dibujar
 * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical
ni horizontal)
 * Parámetro 9: Ponerlo en la capa superior */
ScreenManager.SpriteBatch.Draw(lanzador.sprite, lanzador.posicion,
null, Color.White, lanzador.rotacion, lanzador.centro, 1f,
SpriteEffects.None, 0);
```

Código 6. Dibujo del movimiento de un sprite

➢ Detección de colisiones

La detección de colisiones es uno de los aspectos más importantes de cualquier juego. Un mal sistema de detección de colisiones puede acabar con la mecánica de un juego, especialmente en juegos en los que la velocidad de movimientos es un factor determinante. Hay diferentes formas de manejar la detección de colisiones y la elección del método correcto depende de muchos factores. Por ejemplo, si el juego es 2D o 3D o si es un juego de carreras, aventuras, etc.

Para el caso específico de los juegos en 2D, aunque hay diferencias entre unos estilos de juego y otros, existen tres métodos fundamentales para detección de colisiones: detección de colisión con figuras geométricas, detección por celdas y detección por píxel. Para este juego se ha optado por las figuras geométricas, utilizando rectángulos para ello como se muestra en la siguiente figura:

Figura 82. Rectángulo de colisiones

El tipo de detección de colisiones con rectángulos es el proceso de analizar si existen objetos que están chocando, y en el caso de que los haya, indicar donde ocurrió la colisión. Generalmente, el proceso de detección de colisiones conlleva los siguientes pasos:

1. Determinar qué objetos deben ser analizados.
2. Analizar si ocurrió una colisión entre los objetos.
3. Mostrar la posición en que ocurrió la colisión.
4. Generar una respuesta a la colisión.

Una solución para poder identificar si dos objetos están colisionando, es a través de los **Bounding Boxes**.

Un Bounding Box es un rectángulo que se crea para los sprites, generalmente del mismo, o de un tamaño algo menor de la textura del sprite, que sirve para poder compararlo con los demás rectángulos de Bounding Box de los demás sprites.

Este enfoque para la detección de colisiones es muy simple de realizar en XNA, puesto que la clase **Rectangle** contiene un método llamado *Intersects*, el cual pide como parámetros dos rectángulos para compararlos, y un tercer rectángulo para escribir en él la información de un nuevo rectángulo que se crea en el área de los dos rectángulos que hicieron colisión. Si la comparación de los dos rectángulos dio como resultado que no existe colisión alguna, entonces el método *Rectangle.Intersect* entregará un *Rectangle.Empty*.

El modo de usar los Bounding Boxes en el juego se muestra en el siguiente código:

```
rectangle = new Rectangle((int)posicion.X, (int)posicion.Y,
sprite.Width, sprite.Height);
bbox = new BoundingBox(new Vector3(posicion, 0), new
Vector3(posicion.X + sprite.Width, posicion.Y + sprite.Height, 0));
```

Código 7. Detección de colisiones

El código muestra la inicialización de la detección de colisiones de un objeto de la clase ObjetosJuego y se basa en dos instrucciones, *rectangle* y *bbox*.

La primera genera el rectángulo que va a contener el objeto, para ello se le pasa como parámetros la posición del objeto, la anchura y la altura. La segunda crea 2 objetos **Vector3** que son el punto mínimo y el punto máximo del objeto.

Respecto a los métodos, existen dos métodos que se usan para realizar la detección que son los siguientes:

- *Contains*: Este método dice si un objeto contiene a otro, si se encuentra dentro de su superficie.
- *Intersects*: Este método dice si dos objetos han colisionado. El método intersects toma como parámetro otro objeto con un BoundingBox inicializado. En caso de que se superpongan, el método devuelve un valor verdadero.

El siguiente código representa la decisión de lo que hacer cuando un cohete es interceptado por una bola que está representada en la matriz de objetos.

```
if (MatrizBolas[f, c].bbox.Intersects(cohete.bbox) && MatrizBolas[f,
c].esta_activo) //El cohete colisiona con un objeto de la matriz
{
    EnviarCoheteAMatriz(f, c); //Posicionamos el cohete en la matriz
    romperBucle = true;
    break;
}
```

Código 8. Intercepción de objetos BoundingBox

La siguiente figura muestra una bola azul (cohete) que es interceptada por una bola de color celeste de la matriz de objetos. Cuando esto ocurre, el método EnviarCoheteAMatriz del código anterior, sitúa el cohete en una nueva posición de la matriz, y según el centro de masas donde haya colisionado, la coloca a su derecha, su izquierda, debajo-derecha o debajo-izquierda. En este caso, al colisionar con la parte de debajo de la bola y a su derecha, la coloca en esa posición.

Figura 83. Colisión de dos bolas

La figura 84 representa las divisiones que se toman para decidir dónde se coloca una bola que ha colisionado. Es decir, si se divide una bola en 4 cuadrantes, cuando un cohete colisiona con dicha bola se estudia con qué cuadrante ha colisionado, decidiendo de esta manera el lugar que ocupará dentro de la matriz. Si colisiona en el primer cuadrante, el cohete se situará a la misma altura que la bola colisionada, a su izquierda; si la colisión se produce dentro del segundo cuadrante, el cohete se situará a la derecha de la bola; si es en el tercer cuadrante, el cohete quedará situado justo debajo de la bola colisionada y a la izquierda, y si como es el caso del ejemplo anterior la colisión se produce en el cuarto cuadrante, el cohete se sitúa debajo y a la derecha de la bola colisionada.

Figura 84. División del centro de masas de una bola

➢ Matriz de objetos

La matriz de objetos representa las bolas que el jugador debe destruir. La matriz está formada por 17 filas y 21 columnas y todos los objetos que contienen son del mismo tipo: *ObjetosJuego*

La matriz inicial la genera el método *GenerarPantallaInicial* y su cometido principal consiste en inicializar la matriz con los elementos necesarios para comenzar la partida, que en este caso está formada por las 17 filas y 8 columnas. Según la fila y/o columna sea par o impar, la posición de la bola en la pantalla variará de manera que las bolas no queden exactamente unas debajo de otras, sino que haga un efecto de *zigzag*.

Una bola mide exactamente 30x30 *pixeles*[36], por lo que si una fila impar comienza en la posición 0, la fila par comenzará en la posición 15, para que se produzca este efecto.

Para inicializar la pantalla, se generan las texturas de las bolas y se cargan los atributos para la detección de colisiones. La única diferencia estriba en las bolas que se ven en pantalla y las que no. Inicialmente se verán 8 columnas de bolas, que se ven ya que en la inicialización el parámetro *esta_activo* es verdadero, y que a su vez son susceptibles de ser colisionadas, no como las bolas cuyo parámetro esta_activo cuando es falso, que no cuentan para la detección de colisiones. El siguiente código muestra la inicialización de la matriz de objetos:

```
for (fila = 0; fila < MAXFILAS; fila++)
{
   PosY = (float)fila * 30f;
   if (fila % 2 == 0)
      PosX = 0f;
   else
      PosX = 15f;
   for (numBolas = 0; numBolas < MAXCOLUMNAS; numBolas++)
   {
      colorBolaInicial = GenerarColorBola();
      System.Threading.Thread.Sleep(ContTiempoNivel);

      textBolaInicial = GenerarTexturaBola(colorBolaInicial);
      MatrizBolas[fila, numBolas] = new ObjetosJuego(textBolaInicial,
colorBolaInicial);
      if (fila < FilasGeneradas)
         MatrizBolas[fila, numBolas].esta_activo = true;
      else
         MatrizBolas[fila, numBolas].esta_activo = false;
      MatrizBolas[fila, numBolas].posicion.X = PosX;
      MatrizBolas[fila, numBolas].posicion.Y = PosY;
      //Cargamos los atributos para controlar la detección de colisiones
      MatrizBolas[fila, numBolas].rectangle = new
Rectangle((int)MatrizBolas[fila, numBolas].posicion.X,
(int)MatrizBolas[fila, numBolas].posicion.Y, MatrizBolas[fila,
numBolas].sprite.Width, MatrizBolas[fila, numBolas].sprite.Height);
      MatrizBolas[fila, numBolas].bbox = new BoundingBox(new
Vector3(MatrizBolas[fila, numBolas].posicion, 0), new
Vector3(MatrizBolas[fila, numBolas].posicion.X + MatrizBolas[fila,
numBolas].sprite.Width, MatrizBolas[fila, numBolas].posicion.Y +
MatrizBolas[fila, numBolas].sprite.Height, 0));
      PosX += 30f;
   }
}
```

Código 9. Inicialización de la matriz de objetos

➢ Teclado, ratón y efectos de sonido

Para finalizar el punto dedicado a los elementos que intervienen en el juego hay que destacar las entradas del juego por un lado, que son el teclado y el ratón, y los efectos sonoros, que aunque la implementación sea muy sencilla, hay que destacar la importancia que se merece en el juego, ya que mucha parte del éxito de los juegos viene determinada por los efectos sonoros y la banda sonora.

- **Entradas (inputs)**

  Las entradas del juego vienen definidas por el teclado y el ratón. Dentro del menú principal, el jugador se mueve a través de las opciones gracias a la clase *InputState*, que hereda de *ScreenManager*, la cual tiene implementadas las propiedades y métodos necesarios para que el jugador se mueva por el menú y seleccione las opciones que desee.

  El jugador puede moverse de una opción a otra con las flechas de **Arriba** y **Abajo**, y para ejecutar la opción seleccionada se hace pulsando la tecla **Enter**. Dentro de las subpantallas se puede volver hacia la anterior pulsando **Esc**. En las opciones de "*Nivel de Dificultad*" y "*Mejores puntuaciones*" hay que seleccionar uno de los tres niveles del juego pulsando las teclas "**1**", "**2**" ó "**3**" según corresponda.

  Dentro de la partida, las entradas para el movimiento del lanzador están definidas por las flechas **Izquierda** y **Derecha** por un lado, y el movimiento de izquierda a derecha del ratón. La **barra espaciadora** se destina para el disparo de los cohetes.

  Anteriormente en el código 3 se ha explicado cómo se realiza el movimiento del lanzador a través del teclado; en el siguiente código se muestra la misma acción realizada a través del ratón:

```
MouseState Raton = Mouse.GetState();//Código para leer el ratón
if (Raton != anteriorRaton) //El lanzador cambia de ángulo
{
   int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la
anterior posición del ratón y la nueva.
   lanzador.rotacion += (float)DiferX / 100; //El movimiento en X del
ratón cambia el ángulo del lanzador
}
if (Raton.LeftButton == ButtonState.Pressed &&
!(anteriorRaton.LeftButton == ButtonState.Pressed)) DispararCohete();
anteriorRaton = Raton; //Se actualiza el estado del ratón

//Esta instrucción limita entre los grados el giro del lanzador.
lanzador.rotacion = MathHelper.Clamp(lanzador.rotacion, -MathHelper.Pi
/ 2.5f, MathHelper.Pi / 2.5f);
```

Código 10. Movimiento del lanzador a través del ratón

- **Efectos de sonido**

  El juego tiene cuatro efectos de sonido que son los siguientes:

  - **Disparar**. Efecto producido cuando un cohete es disparado desde el lanzador.
  - **Posicionar**. Se produce cuando un cohete se posiciona en la matriz de objetos.
  - **EliminarBola**. Este efecto reproduce la eliminación de una bola de la matriz.
  - **Explosionar**. Efecto producido cuando el jugador limpia la pantalla de bolas.

Los objetos que reproducen los sonidos vienen definidos por el tipo *SoundEffect* de la clase **Microsoft.Xna.Framework.Audio**. La reproducción del efecto sonoro se realiza a través del método *Play* del objeto creado. El siguiente código representa la carga de los efectos de sonido que contiene en juego:

```
//Cargamos los efectos de sonido
DisparарBola =
ScreenManager.Content.Load<SoundEffect>("Sonidos/Disparar");
EliminarBola =
ScreenManager.Content.Load<SoundEffect>("Sonidos/EliminarBola");
PosicionarBola =
ScreenManager.Content.Load<SoundEffect>("Sonidos/Posicionar");
LimpiarPantalla =
ScreenManager.Content.Load<SoundEffect>("Sonidos/Explosion");
```

Código 11. Definición de objetos de efectos de sonido

3.3.2. Algoritmos

Este es uno de los puntos más creativos del juego, ya que se va a explicar el *algoritmo*[37] desarrollado e implementado para determinar las bolas que deben ser eliminadas tras una detección de colisiones, que por otro lado es la parte más complicada del juego. Durante este apartado y por la naturaleza del algoritmo a las bolas las llamaremos nodos.

El algoritmo está dividido en tres partes:

1. **Búsqueda**. Encontrar nodos adyacentes al cohete de manera recursiva a través de árboles de búsqueda de aquellas que sean del mismo color del cohete.
2. **Destrucción**. Si el número de nodos encontrados es mayor de tres, se procede a eliminar todos ellos.
3. **Actualización**. Consiste en destruir todos aquellos nodos o grupos de nodos que tras el paso de destrucción quedan "*colgados*" en la matriz de objetos, es decir, todos aquellos nodos de un determinado nivel de la matriz, en el que no se consigue encontrar un camino a través de nodos adyacentes que hagan llegar al nivel superior de la matriz.

A continuación se pasa a explicar cada uno de los tres puntos en detalle.

- **Fase de Búsqueda**

Cuando un cohete colisiona con una bola de la matriz de objetos se realizan los siguientes pasos:

1. Posicionar el cohete en la celda de la matriz que le corresponde y añadirlo a la lista general del algoritmo.
2. Buscar a través de un algoritmo recursivo todos los nodos adyacentes del mismo color al nodo del cohete posicionado en la matriz.
3. Si encuentra tres ó mas nodos del mismo color, incluido el nodo del cohete, los elimina de la matriz.

El algoritmo de búsqueda consiste en encontrar en la matriz de objetos todos los nodos adyacentes entre sí al nodo del cohete de igual color que éste. El objetivo es generar un árbol donde cada nodo llama recursivamente al algoritmo para obtener la lista de nodos adyacentes a sí mismo que son del mismo color al cohete.

El algoritmo de búsqueda propiamente dicho lo realiza el método **ActualizarEstadoBolas**. Este método recibe dos listas dinámicas y el color del nodo del cohete que ha sido posicionado. A través del algoritmo de búsqueda almacenará en las listas los objetos correspondientes a las bolas adyacentes del mismo color que el cohete.

Las listas dinámicas están formadas por objetos del tipo **posicion**, que contiene la posición X e Y de las bolas de la matriz de objetos susceptibles de ser eliminadas. Cuando el algoritmo es llamado por primera vez, ambas listas dinámicas contienen un único objeto o nodo, que corresponde a la posición que ocupa el nodo del cohete en la matriz de objetos.

La lista *ALIntercepcionesActuales* contiene una lista en forma de árbol de las posiciones de las bolas adyacentes al nodo del que se están realizando las comparaciones; esta lista se limpia en cada llamada recursiva. La lista *ALGeneral* contiene la suma de todas la listas *ALIntercepcionesActuales* que se van generando en cada llamada recursiva. Las llamadas recursivas se realizan mientras existan nodos que se puedan comparar. El siguiente código muestra el algoritmo empleado.

```
private void ActualizarEstadoBolas(ref ArrayList ALGeneral,
ArrayList ALIntercepcionesActuales, string ColorBola)
{
   ALGeneral.Add(ALIntercepcionesActuales); //Se almacena el
array con los nodos actuales al array general. El primer
elemento del array es el nodo raiz (cohete)
   ArrayList ALNuevasIntercepciones = new ArrayList(); //En este
array se almacenan los elementos hijos del array actual

   for (int i = 0; i < ALIntercepcionesActuales.Count; i++)
   {
      int pFila = ((posicion)ALIntercepcionesActuales[i]).pFila;
      int pCol = ((posicion)ALIntercepcionesActuales[i]).pCol;

      if (pFila % 2 == 0) //Estamos en una fila par
         ALNuevasIntercepciones =
CompararFilaPar((posicion)ALIntercepcionesActuales[i],
ColorBola);
      else //Estamos en una fila impar
         ALNuevasIntercepciones =
CompararFilaImpar((posicion)ALIntercepcionesActuales[i],
ColorBola);

//Si hemos encontrado más nodos hijo, lo añadimos al array
genereal y llamamos de nuevo al método actual para tratar los
nuevos nodos
      if (ALNuevasIntercepciones.Count > 0)
      {
         TotalInterceptados += ALNuevasIntercepciones.Count;
         ALIntercepcionesActuales = ALNuevasIntercepciones;
         ActualizarEstadoBolas(ref ALGeneral,
ALIntercepcionesActuales, ColorBola);
      }
   }
}
```

Código 12. Algoritmo de búsqueda de nodos

Como se ve en el algoritmo, las comparaciones dependen de la fila en la que se encuentre el nodo, esto se debe a la posición de las bolas en la pantalla, ya que como están en forma de zigzag, el control de las bolas de alrededor al nodo que se está comparando y del perímetro es distinto en filas pares que impares. Por ejemplo, para realizar comparaciones de un nodo que esté en una fila par y sea el primero de la fila (posición [2,0]), se podrá comparar sólo con una bola del nivel anterior (posición [1,0]) ya que la siguiente (posición [1,1]) no es

adyacente; sin embargo, un nodo correspondiente a una bola de una fila impar (posición [3,0]) puede compararse con dos bolas del nivel anterior (posiciones [2,0] y [2,1]) ya que SI son adyacentes. La siguiente figura muestra este concepto, siendo la bola amarilla recuadra en rojo la bola de la fila par (fila 2) y la morada recuadrada en rojo la bola de la fila impar (fila 3). Las filas y columnas de la matriz empiezan por la posición [0,0].

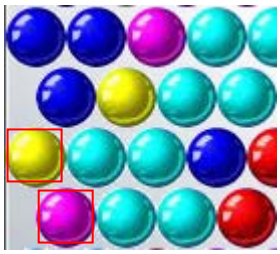

Figura 85. Comparación de bolas adyacentes

Durante las fases de comparación, el algortimo busca nodos del mismo color al nodo que se está comparando. El orden siempre es empezando por los nodos del nivel inmediatamente superior hacia abajo y de izquierda a derecha siempre que sean adyacentes al nodo que se está comparando. Para comprender mejor el algoritmo se expone el ejemplo de la figura 86.

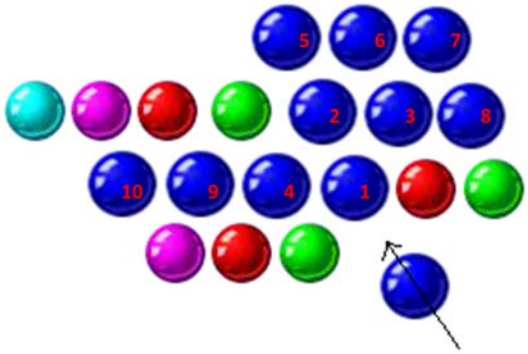

Figura 86. Supuesto de una colisión

Como se puede ver, la bola de color azul que dirige la flecha va a colisionar con la bola de color azul número 1. Al colisionar, el nodo que impacta se añade a la lista y busca nodos adyacentes del mismo color que él según el orden señalado anteriormente, es decir, primero encima suyo a la izquierda, encontrando el nodo número 1; a continuación a su derecha adyacente la bola roja. Como la bola verde siguiente no es adyacente, no la compara. A continuación, en su mismo nivel a la izquierda encuentra la bola verde, y como no hay nodos a su derecha y debajo, termina la búsqueda de este nodo.

En este punto la lista *ALGeneral* contiene en su primera casilla (casilla 0) el nodo que intercepta, suponiendo que sea la posición [10,8] y en la casilla 1 las posiciones de la primera comparación, es decir, sería la posición [9,7].

Como se han encontrado nodos del mismo color, se vuelve a llamar a la función y realiza la misma búsqueda, encontrando para el nodo 1 los nodos 2 ,3 y 4 que estarían recogidos en la lista *ALIntercepcionesActuales* y se añadiría a la casilla 2 de *ALGeneral*.

El nodo 2 por tanto encontraría en su comparación el 5 y el 6. El nodo 3 encontraría el nodo 6 y 7, pero como el 6 ya está en la lista, lo obvia, al igual que pasa con el nodo 9, que encuentra los nodos 10 y 4, pero como el 4 ya está marcado, tampoco lo añade.

Por tanto, la lista ALGeneral quedaría de la siguiente manera:

Casilla 0:	([10,8])	**Cohete**
Casilla 1:	([9,7])	**Nodo 1**
Casilla 2:	([8,7], [8,8], [9,6])	**Nodos 2, 3 y 4**
Casilla 3:	([7,6], [7,7]), ([7,8], [8,9]), ([9,5])	**Nodos (5y 6), (7 y 8), 9**
Casilla 4:	([9,4])	**Nodo 10**

Tabla 2. Lista ALGeneral

Al llegar al nodo 10, como ya no hay más nodos iguales que comparar, se llegaría a la salida del algoritmo.

Otros factores que se han tenido en cuenta durante la fase de búsqueda son los siguientes:

- Gestión de los nodos de laprimera fila de la matriz, ya que no se pueden comparar con nodos que haya por encima de ellos.
- Gestión de los nodos que lleguen al límite máximo de filas de la matriz, para no comprarar con nodos que haya por debajo.
- Gestión de los primeros nodos de columna, para no comprarlos con nodos por su izquierda.
- Gestión de los últimos nodos de columna, para no compararlos con nodos por su derecha.

Una vez que se tiene el árbol de nodos susceptibles de ser eliminados, hay que proceder a la eliminación de éstos.

- Fase de Destrucción

Cuando el algoritmo de búsqueda termina, se ha generado una lista con todos los nodos que se deben eliminar. Para ello se recorre la lista generada y se inicializan las posiciones en la matriz de objetos para que no aparezcan en la pantalla, y para que los nodos eliminados no sean susceptibles de colisión.

A la misma vez que se realiza esta operación se van añadiendo los puntos generados por la eliminación de las bolas. El siguiente código muestra la fase de Destrucción, siendo *ALBolasInterceptadas* el array *ALGeneral* visto anteriormente que contiene la lista de los nodos que han de ser eliminados.

```
if (TotalInterceptados > 2) //En este caso cumple el requisito para
eliminar las bolas interceptadas
{
   int NumBolas = 0;
   for (int i = 0; i < ALBolasInterceptadas.Count; i++)
   {
      ArrayList AL = new ArrayList();
      AL = ((ArrayList)ALBolasInterceptadas[i]);
      for (int j = 0; j < AL.Count; j++)
      {
         NumBolas++;
         int pFila = ((posicion)AL[j]).pFila;
         int pCol = ((posicion)AL[j]).pCol;
         //La bola no se ve en pantalla
         MatrizBolas[pFila, pCol].esta_activo = false;
         //Eliminamos la detección de colisiones
         MatrizBolas[pFila, pCol].bbox = new BoundingBox();
         MatrizBolas[pFila, pCol].rectangle = new Rectangle();
         //Desmarcamos la marca de eliminacion para futuras
comparaciones de otra bola en esa posición
         MatrizBolas[pFila, pCol].marcaElim = false;
         //Damos más puntos cuantas más bolas se intercepten
         if (NumBolas > 3)
            Puntos = Puntos + ((NumBolas / 2) * 10);
         else
            Puntos += 10;
         EliminarBola.Play();
      }
   }
}
```

Código 13. Destrucción de nodos

➢ Fase de Actualización

Una vez destruidas las bolas y añadidos los puntos conseguidos por ello, hay que proceder a actualizar el juego, ya que cabe la posibilidad de que haya nodos que se hayan quedado "*colgados*" tras la destrucción de los nodos; por ello, la función **ActualizarEstadoJuego** se encarga de esto.

El algoritmo empleado para actualizar el juego y eliminar las bolas o grupos de bolas que quedan en el aire consiste en recorrer la matriz de objetos y buscar para cada elemento, empezando por la segunda fila de la matriz (fila 1), un camino que le llegue a la fila inmediatamente anterior de la matriz pasando siempre por bolas que estén activas. De esta manera queda asegurada la actualización de la matriz y en la pantalla no quedan bolas en el aire. En el código 14 se puede ver el algoritmo empleado en este caso.

```
int fila; int col; bool FilaLimpia = false;

for (fila = 1; fila < FilasGeneradas; fila++)
{
   FilaAnterior = fila - 1; //El nodo tiene que llegar a la fila anterior
para no ser eliminado
   bool EliminarFila = true;
   for (col = 0; col < MAXCOLUMNAS; col++)
   {
      if (MatrizBolas[fila, col].esta_activo == true)
      {
         FilaAntEncontrada = false;
         ALListaPendElim = new ArrayList();
         posicion pos = new posicion();
         pos.pFila = fila;
         pos.pCol = col;
         MatrizBolas[fila, col].marcaElim = true; //Marcamos el primer
nodo como pendiente de eliminar
         ALListaPendElim.Add(pos);
         BuscarFilaAnterior(pos);
         if (!FilaAntEncontrada)
            EliminarBolasEnAire();
         else
            DesmarcarPendElim();
      }
      //Comprobamos si sigue activa esa posicion para chequear el estado
de filas generadas
      if (MatrizBolas[fila, col].esta_activo == true && EliminarFila ==
true)
         EliminarFila = false;
   }
   if (EliminarFila == true)
      FilasGeneradas--;
}
```

Código 14. Actualización del juego

Una vez que el juego ha quedado actualizado, ya está preparado para un nuevo disparo y repetir el ciclo explicado en esta sección.

3.3.3. Sistema de puntuación

El sistema de puntuación empleado se basa en sumar puntos por bolas destruidas, premiando con más puntos cuantas más bolas se consigan destruir a la vez. La forma de conseguir puntos consiste por tanto en explotar el máximo número de bolas agrupadas del mismo color, siendo el sistema de puntuación el siguiente:

1 bola = 10 puntos, siendo 3 bolas el mínimo exigido y por tanto 30 puntos.
Por cada bola adicional superando las 3 bolas se sigue la fórmula siguiente:

(Numero de Bola / 2) x 10;

Por tanto, si se explotan 7 bolas juntas, los puntos conseguidos serían **(3x10) + (4/2)x10 + (5/2)x10 + (6/2)x10 + (7/2)x10**. En total **130 puntos**.
Por cada nivel superado, se asignan puntos, siendo éstos el número de nivel superado multiplicado por cien, es decir, si se supera el nivel 3, se asginan **3x10 = 30 puntos.**
Adicionalmente, si el jugador es capaz de limpiar el mapa de bolas y quedarse sin bolas en la pantalla, se le compensará con **10000** puntos extra.

3.3.4. Niveles

El juego consta de dos tipos de niveles que son por un lado el nivel de dificultad al comenzar una partida, y por otro lado el nivel de dificultad que el jugador va obteniendo tras superar hitos durante el juego.

Antes de comenzar una partida el jugador selecciona el nivel de dificultad en el que desea jugar, siendo las opciones FACIL, MEDIO, DIFICIL.

La característica principal en esta elección se basa en 2 conceptos:

1. Un nivel fácil genera una pantalla inicial con grupos más juntos de bolas del mismo color, generando en el nivel difícil una pantalla con bolas de colores más dispersas entre sí.
2. En el nivel fácil, el intervalo de tiempo que tarda en generarse una nueva fila es mayor que en el nivel difícil, generándose una nueva fila cada 40 segundos en el nivel fácil, cada 30 en el nivel medio, y cada 20 segundos en el nivel difícil.

Como se ha dicho, durante la partida el jugador va consiguiendo puntos, y cuando supera múltiplos de 1000 puntos la partida aumenta un nivel, provocando que el intervalo de tiempo que tarda en generarse una nueva fila disminuya un segundo por nivel alcanzado, es decir, que una partida en nivel DIFICIL y que el jugador haya llegado al nivel 5, el tiempo entre nuevas filas de bolas es de 20-5 = 15 segundos.

Capítulo 4

Objetivos

En esta sección se comentarán los objetivos pretendidos y los objetivos logrados durante el desarrollo del proyecto.

Se analizará el resultado teniendo en cuenta los objetivos y las expectativas que se fijaron en un principio respecto a los problemas que han surgido.

Finalmente se comentarán las conclusiones extraídas tras la realización de este proyecto.

4.1. Objetivos y problemas encontrados

Antes de comenzar este proyecto, había una serie de requisitos que había que cumplir para poder realizarlo con garantías, es decir, se necesitaba una base que garantizase que el alumno tuviese la capacidad de poder realizarlo con cierta soltura, sin embargo, a lo largo del desarrollo del proyecto han surgido muchas dudas y problemas, unos con los que ya se contaba, y otros que fueron surgiendo según avanzaba el desarrollo.

Las tres características que el alumno debía cumplir eran las siguientes:

- Haber cursado Ingeniería Técnica en Informática de Gestión.
- Conocer el entorno de programación C#.
- Ilusión por conocer y trabajar el mundo de los videojuegos.

Aun cumpliendo los tres requisitos, enfrentarse a un nuevo reto en el que el desconocimiento de todo lo relacionado con los videojuegos era total, agregaba un nuevo problema solventado eso sí, con la el último requisito.

Uno de los primeros problemas a solventar era aprender funcionalmente cómo desarrollar un videojuego.

Los objetivos pretendidos por tanto radicaban en los siguientes términos:

- Familiarización con el entorno de programación de videojuegos.
- Conocer la API *XNA* proporcionada por *Microsoft*.
- Establecer ventajas e inconvenientes respecto a otros productos del mercado.
- Desarrollar un clon del juego Bubble shooter, añadiéndole nuevas funcionalidades pero que reflejase fielmente la esencia del juego con la ayuda de lo anteriormente aprendido.

Los principales problemas encontrados al inicio del proyecto fueron puramente funcionales y conceptuales, por lo que había que tener claros los siguientes conceptos:

- Qué son los sprites.
- Cómo mover objetos por la pantalla.
- Manejo de las entradas por teclado y ratón.
- Gestión de la detección de colisiones.
- Creación de los menús.

- Gestión de la navegación por pantallas.

En lo que se refiere a las características del juego, el mayor problema ha sido el desarrollo de los algoritmos, ya que en una primera versión, cuando un cohete colisionaba con una bola de su mismo color conseguía hacerla explotar, pero no así con las de alrededor.

La segunda versión consistió en arreglar esta circunstancia, y al disparar una bola y colisionar con otra de su mismo color, conseguía hacer explotar todas las adyacentes del mismo color, sin embargo al actualizar la pantalla, en ocasiones quedaban bolas en el aire sin colgar de ninguna otra, por lo que en la tercera y última versión se creó un algoritmo adicional para eliminar las bolas que quedaban en el aire.

Una vez finalizado el desarrollo del videojuego se puede decir por tanto que los objetivos pretendidos han quedado cumplidos satisfactoriamente.

4.2. Conclusiones

Una vez finalizado el proyecto, se puede afirmar que el trabajo realizado ha cumplido de manera satisfactoria la mayoría de los objetivos que se habían fijado. En primer lugar, hay que destacar que uno de los factores más importantes para llevar a cabo un desarrollo de este tipo de aplicaciones, es que están muy vinculadas a las comunidades que le dan soporte. Esto es algo donde el uso de *XNA* ha sido una ventaja respecto a otras plataformas. Por lo tanto, se puede recalcar que *XNA* está respaldado por *Microsoft* y una comunidad enorme que le colabora en foros y blogs.

En cuanto al propio videojuego, sin duda se han cubierto todos los objetivos y requisitos planteados a lo largo del desarrollo. El clon del *Bubble shooter* refleja fielmente la esencia del original en cuanto al objetivo del juego, el manejo del lanzador, los efectos de sonido, sistema de puntuaciones, etc. Se le han añadido nuevas funcionalidades como las poder usar el lanzador con el ratón o el teclado, un sistema de puntuación que premia la explosión de mayores bolas agrupadas, y dificultar la partida dotándola de mayor rapidez a medida que el jugador supera niveles.

En lo referente a la programación, el API *XNA* proporciona un gran número de soluciones a cada problema que se ha planteado. En el caso de este proyecto, los primeros pasos se llevaron a cabo con los movimientos de las bolas y su posterior animación. En este aspecto el API es muy completo proporcionando acceso a dispositivos de teclado, ratón y mandos de *XBOX 360* (con librerías disponibles en la comunidad *Creator's Club* es posible emular los mandos aunque no sean los de *XBOX 360*). En cuanto a la animación, *XNA* proporciona soluciones para la carga de texturas 2D así

como de modelos 3D. El siguiente paso fue el manejo de colisiones, aspecto de sobra explicado en este documento. Las funciones matemáticas del API permiten manejar este aspecto sin problemas. La última parte, relacionada con las explosiones, están más relacionadas con las decisiones de diseño tomadas y su implementación. Hay que resaltar un último aspecto relacionado con *XNA*, ya que la dimensión del juego no explota por completo la librería. Faltan por explorar las funciones de juego en 3D, aspecto que queda pendiente para desarrollos futuros.

En lo personal, debo destacar que la realización de este proyecto me ha enriquecido tanto en lo personal como en lo profesional, dándome la oportunidad de conocer una nueva herramienta en un mundo profesional como el de los videojuegos que poco tiene que ver con el mío.

He aprendido las técnicas básicas de la realización de videojuegos, dándome la posibilidad de poder explorar este fascinante mundo y sacar el mayor partido posible a la API *XNA*, que tan buenos juegos es capaz de desarrollar.

Capítulo 5

Líneas futuras

Una vez realizado el proyecto, se analiza en el capítulo 5 los posibles trabajos futuros que se pueden desarrollar a partir de éste.

Se comentarán algunos de los posibles trabajos a llevar a cabo en el futuro y que no se han tratado en este proyecto.

5.1. Futuros trabajos a realizar

En todo proyecto fin de carrera se establecen los objetivos a realizar al comienzo de éste, sin embargo, a medida que se va avanzando suelen surgir nuevas funcionalidades y mejoras que se pueden aportar al proyecto, sin embargo esto haría interminable el proyecto, por lo que una vez superadas todos los objetivos iniciales, algunas de estas mejoras quedan plasmadas en este capítulo para llevarlas a cabo en futuras líneas de trabajo.

En primer lugar, *XNA* permite la función multijugador. Este no es un juego que se caracterice por poder jugarse en red, ya que sólo hay un jugador en la partida, sin embargo, la posibilidad de poder jugar en línea ofrecería la posibilidad de establecer un sistema de puntuación general donde los jugadores que comienzan a la vez una partida, puedan ir viendo las puntuaciones de los demás estableciendo un tiempo máximo de partida.

Otra mejora podría consistir en adaptar este formato de juego para PC, en formato *XBOX 360,* para poder publicarlo y en su caso comercializarlo.

Se puede ir más allá siempre que la imaginación lo permita, y dado que XNA permite desarrollar juegos en 3D, una nueva línea de trabajo podría consistir, basándose en este trabajo, en utilizar las funciones 3D para crear un cubo donde las bolas se almacenen en una matriz tridimensional.

Respecto a las mejoras propiamente dichas de este trabajo, se podría mejorar el sistema de detección de colisiones. En este proyecto se ha llevado a cabo mediante el sistema de rectángulos, sin embargo, para que fuese realmente perfecto podría desarrollarse un nuevo sistema de colisiones donde la figura geométrica sea un círculo en vez de un rectángulo. Otro sistema de colisiones más perfeccionado consiste en la detección de colisiones por pixeles, en caso de que se requiera de un nivel de perfección máximo.

Anexo A

Planificación

En el siguiente anexo se expone el ciclo de vida del proyecto mostrado a través del diagrama de GANTT.

En dicho diagrama se muestran las tareas realizadas, los recursos y el tiempo que ha llevado hacer cada tarea.

A1. Introducción

Todo proyecto profesional debe contener una planificación previa al desarrollo de la misma. Una buena planificación ayuda a conocer los recursos humanos que son necesarios y el momento en el que son necesarios. Por otro lado, en la planificación se estructura el proyecto en fases asignando las tareas y los recursos necesarios en cada fase, lo cual facilita al jefe de proyecto la posibilidad de calcular lo más fielmente posible el plazo de entrega del proyecto. Todo esto en conjunto, ofrece un mayor control sobre el proyecto y una mayor precisión a la hora de calcular su viabilidad, ya que al tener una noción de los plazos y recursos necesarios para llevar a cabo las distintas partes, se tendrá una idea aproximada de su coste total. En primer lugar, antes de detallar las fases del proyecto, se hará una pequeña referencia a las fases típicas de un proyecto comercial, para poder comparar su planificación y desarrollo con el de este proyecto.

A2. Ciclo de vida

En el apartado 2.3 de esta memoria se explica detalladamente el ciclo de vida de un proyecto general, y en particular de un videojuego comercial. A continuación se hace un resumen de las fases más importantes en la creación de un videojuego comercial:

- **Concepción de la idea del videojuego**.

  En esta fase, como su propio nombre indica se define el concepto del juego. A partir de una idea original, se crea una propuesta del jugo y el arte conceptual, que servirá como primera base y ejemplo de cómo serán el juego y su historia. En esta fase se definirán conceptos como el género, las características generales, la ambientación, plataformas de desarrollo, cronograma estimado, presupuesto y análisis de riesgos. En esta fase participan diseñadores y analistas.

- **Diseño.**

  El diseño es la fase en la que se van a detallar todos los elementos y componentes que van a dar forma al juego. En el diseño de un videojuego se han de especificar aspectos como la historia, el arte conceptual y el sonido. En esta fase intervienen diseñadores, analistas y artistas conceptuales.

- **Planificación.**

  En la fase de Planificación se identifican todas las tareas del proyecto, se estima el tiempo que conlleva realizarlas y se reparten al equipo de desarrolladores. Además, se establecen reuniones de seguimiento a los largo del proyecto para verificar si la planificación va por los cauces establecidos. Los jefes de proyecto realizan este trabajo.

- **Producción.**

  En esta fase comienza la construcción del juego. Se escribe el código, se crea el arte gráfico y los sonidos. En esta fase hay que tener en mente que el juego puede cambiar o evolucionar, apareciendo nuevas características o quitando otras, manteniendo siempre actualizada la planificación.

- **Pruebas.**

  Fase en la que los probadores, como su propio nombre indica, prueban el videojuego. A lo largo del desarrollo del videojuego y según se van cumpliendo hitos, los probadores hacen todos los test a cada fase desarrollada. Cuando el desarrollo termina, realizan la prueba final para detectar anomalías o fallos antes de ser lanzado al mercado.

- **Mantenimiento.**

  Después de la distribución surge una nueva fase de testeo, realizada esta vez por los usuarios. En la mayoría de los juegos actuales, se ha hecho casi inevitable la publicación de parches posteriores al lanzamiento, sobre todo en lo que al juego en red se refiere.

A3. Planificación del proyecto

Una vez conocido el ciclo de vida del videojuego, el siguiente paso es definir las fases que componen el proyecto. En este proyecto, se han desarrollado las siguientes fases:

- Fase de formación previa
- Fase estructural del proyecto
- Fase de creación del juego
- Fase de gestión de pantallas
- Fase de elementos extras
- Evaluación y pruebas finales

En primer lugar se definieron una serie de requisitos generales, que representaban aspectos básicos e importantes que el juego debía tener. Posteriormente se centró el proyecto en los elementos del juego (lanzador, base lanzador, cohete y matriz de bolas), controles, animación, movimientos. Una vez implementado se pasó al diseño de niveles. Posteriormente se incluyó la gestión de las puntuaciones máximas y finalmente la gestión de menús y navegación por pantallas.

En todo proyecto existen factores que pueden hacer que la planificación se vea afectada y no se vean cumplidos los plazos iniciales. Este no es un caso aparte y algunas fases han tenido que ser modificadas respecto a la planificación inicial. En este apartado se mostrarán los diagramas de GANTT inicial y el real, y se analizarán los apartados en los que haya habido una desviación en el tiempo y la causa.

Un diagrama de GANTT está compuesto por las tareas que componen el proyecto y el tiempo empleado en ser realizadas por cada recurso. Cada una de las tareas debe tener necesariamente al menos un recurso asignado. La falta de disponibilidad de estos recursos puede provocar que se vea retrasado el comienzo de ciertas tareas, por lo que a menudo, para evitar estos retrasos, se debe aumentar el número de recursos, pudiendo así llevar a cabo múltiples tareas de forma paralela. Además de las tareas anteriormente nombradas a lo largo del proyecto, se llevarán a cabo reuniones periódicas de seguimiento que marcarán los hitos del proyecto.

Las tareas que componen el proyecto, indicado además su duración en días en el diagrama inicial son las que se muestran a continuación en la tabla 3:

Nombre de tarea	Duración
1. Fase de formación previa	**30 días**
1.1. Estudio funcional sobre la creación de videojuegos	10 días
1.2. Formación en XNA	20 días
1.3. Reunión de seguimiento	2 horas
2. Desarrollo de la memoria del proyecto	**245 días**
3. Fase estructural del proyecto	**38 días**
3.1. Análisis de requisitos F.E.	20 días
3.1.1. Establecer los elementos principales	6 días
3.1.2. Establecer los movimientos principales	5 días
3.1.3. Eventos de las pantallas	4 días
3.1.4. Interrupciones del juego	5 días
3.2. Diseño F.E.	7 días
3.3. Implementación F.E.	8 días
3.4. Evaluación F.E.	1 día
3.5. Reunión de seguimiento	2 horas
4. Fase de creación del juego	**105 días**
4.1. Análisis de requisitos F.C.	31 días
4.1.1. Diseño gráfico y creación de los elementos	5 días
4.1.2. Aplicar movimientos a los elementos	10 días
4.1.3. Detección de colisiones	6 días
4.1.4. Algoritmos	5 días
4.1.5. Niveles y puntuaciones	5 días
4.1.6. Reunión de seguimiento	2 horas
4.2. Diseño F.C.	14 días
4.3. Implementación F.C.	35 días
4.3.1. Aplicar movimientos a los elementos	10 días
4.3.2. Algoritmos	15 días
4.3.3. Niveles y puntuaciones	10 días
4.4. Evaluación F.C.	10 días
4.5. Reunión de seguimiento	2 horas
5. Fase de gestión de pantallas	**30 días**
5.1. Análisis de requisitos F.G.	10 días
5.1.1. Diseño gráfico	5 días
5.1.2. Navegación por pantallas	5 días
5.2. Diseño F.G.	5 días
5.3. Implementación F.G.	10 días
5.4. Evaluación F.G.	5 días
5.5. Reunión de seguimiento	2 horas
6. Fase de elementos extras	**27 días**
6.1. Análisis de requisitos F.E.E.	10 días
6.1.1. Efectos de sonido	2 días
6.1.2. Gestión de almacenamiento de puntuaciones	3 días
6.2. Diseño F.E.E.	5 días

6.3. Implementación F.E.E.	10 días
6.4. Evaluación F.E.E.	2 días
6.5. Reunión de seguimiento	2 horas
7. Evaluación y pruebas finales del proyecto	**10 días**
8. Reunión final del proyecto	**2 horas**

Tabla 3. Tareas del proyecto inicial

La tabla 4 que se muestra a continuación representa la duración real de las tareas planificadas al inicio del proyecto.

Nombre de tarea	Duración
1. Fase de formación previa	**30 días**
1.1. Estudio funcional sobre la creación de videojuegos	10 días
1.2. Formación en XNA	20 días
1.3. Reunión de seguimiento	2 horas
2. Desarrollo de la memoria del proyecto	**275 días**
3. Fase estructural del proyecto	**40 días**
3.1. Análisis de requisitos F.E.	20 días
3.1.1. Establecer los elementos principales	6 días
3.1.2. Establecer los movimientos principales	5 días
3.1.3. Eventos de las pantallas	4 días
3.1.4. Interrupciones del juego	5 días
3.2. Diseño F.E.	7 días
3.3. Implementación F.E.	10 días
3.4. Evaluación F.E.	1 día
3.5. Reunión de seguimiento	2 horas
4. Fase de creación del juego	**105 días**
4.1. Análisis de requisitos F.C.	40 días
4.1.1. Diseño gráfico y creación de los elementos	5 días
4.1.2. Aplicar movimientos a los elementos	10 días
4.1.3. Detección de colisiones	10 días
4.1.4. Algoritmos	10 días
4.1.5. Niveles y puntuaciones	5 días
4.1.6. Reunión de seguimiento	2 horas
4.2. Diseño F.C.	14 días
4.3. Implementación F.C.	40 días
4.3.1. Aplicar movimientos a los elementos	10 días
4.3.2. Algoritmos	20 días
4.3.3. Niveles y puntuaciones	10 días
4.4. Evaluación F.C.	10 días
4.5. Reunión de seguimiento	2 horas
5. Fase de gestión de pantallas	**30 días**
5.1. Análisis de requisitos F.G.	10 días
5.1.1. Diseño gráfico	5 días
5.1.2. Navegación por pantallas	5 días

5.2. Diseño F.G.	5 días
5.3. Implementación F.G.	10 días
5.4. Evaluación F.G.	5 días
5.5. Reunión de seguimiento	2 horas
6. Fase de elementos extras	**34 días**
6.1. Análisis de requisitos F.E.E.	5 días
6.1.1. Efectos de sonido	2 días
6.1.2. Gestión de almacenamiento de puntuaciones	3 días
6.2. Diseño F.E.E.	5 días
6.3. Implementación F.E.E.	10 días
6.4. Evaluación F.E.E.	2 días
6.5. Reunión de seguimiento	2 horas
7. Evaluación y pruebas finales del proyecto	**10 días**
8. Reunión final del proyecto	**2 horas**

Tabla 4. Tareas del proyecto real

A continuación se muestra en el diagrama de GANTT inicial las tareas de comienzo y fin correspondientes; además de los recursos asignados a cada una de ellas antes del comienzo del proyecto.

Diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
1	**Fase de formación previa**	**30 días**	**lun 21/09/09**	**vie 30/10/09**	
2	Estudio funcional sobre la creación de videojuegos	10 días	lun 21/09/09	vie 02/10/09	Programador[90%]
3	Formación en XNA	20 días	lun 05/10/09	vie 30/10/09	Programador[90%]
4	Reunión de seguimiento	2 horas	vie 30/10/09	vie 30/10/09	Analista;Jefe Proyecto;Programador
5	Desarrollo de la memoria del proyecto	245 días	lun 02/11/09	vie 08/10/10	Programador[10%]
6	**Fase estructural del proyecto**	**38 días**	**lun 02/11/09**	**mié 23/12/09**	
7	**Análisis de requisitos F.E.**	**20 días**	**lun 02/11/09**	**vie 27/11/09**	
8	Establecer los elementos principales	6 días	lun 02/11/09	lun 09/11/09	Analista
9	Establecer los movimientos principales	5 días	mar 10/11/09	lun 16/11/09	Analista
10	Eventos de las pantallas	4 días	mar 17/11/09	vie 20/11/09	Analista
11	Interrupciones del juego	5 días	lun 23/11/09	vie 27/11/09	Analista
12	Diseño F.E.	7 días	lun 30/11/09	mar 08/12/09	Analista
13	Implementación F.E.	8 días	mié 09/12/09	vie 18/12/09	Programador[90%]
14	Evaluación F.E.	1 día	mié 23/12/09	mié 23/12/09	Probador
15	Reunión de seguimiento	2 horas	mié 23/12/09	mié 23/12/09	Jefe Proyecto;Analista
16	**Fase de creación del juego**	**105 días**	**lun 11/01/10**	**vie 04/06/10**	
17	**Análisis de requisitos F.C.**	**31 días**	**lun 11/01/10**	**lun 22/02/10**	
18	Diseño gráfico y creación de los elementos	5 días	lun 11/01/10	vie 15/01/10	Artista gráfico
19	Aplicar movimientos a los elementos	10 días	lun 18/01/10	vie 29/01/10	Analista
20	Detección de colisiones	6 días	lun 01/02/10	lun 08/02/10	Analista
21	Algoritmos	5 días	mar 09/02/10	lun 15/02/10	Analista
22	Niveles y puntuaciones	5 días	mar 16/02/10	lun 22/02/10	Analista
23	Reunión de seguimiento	2 horas	lun 22/02/10	lun 22/02/10	Jefe Proyecto;Analista
24	Diseño F.C.	14 días	mar 23/02/10	vie 12/03/10	Analista
25	**Implementación F.C.**	**35 días**	**mar 23/02/10**	**lun 12/04/10**	**Programador[90%]**
26	Aplicar movimientos a los elementos	10 días	mar 23/02/10	lun 08/03/10	
27	Algoritmos	15 días	mar 09/03/10	lun 29/03/10	
28	Niveles y puntuaciones	10 días	mar 30/03/10	lun 12/04/10	
29	Evaluación F.C.	10 días	mar 13/04/10	lun 26/04/10	Probador
30	Reunión de seguimiento	2 horas	lun 26/04/10	lun 26/04/10	Jefe Proyecto;Analista
31	**Fase de gestión de pantallas**	**30 días**	**mié 28/04/10**	**mar 08/06/10**	
32	**Análisis de requisitos F.G.**	**30 días**	**mié 28/04/10**	**mar 08/06/10**	
33	Diseño gráfico	5 días	mié 28/04/10	mar 04/05/10	Artista gráfico
34	Navegación por pantallas	5 días	mié 05/05/10	mar 11/05/10	Analista;Artista gráfico
35	Diseño F.G.	5 días	mié 12/05/10	mar 18/05/10	Analista
36	Implementación F.G.	10 días	mié 19/05/10	mar 01/06/10	Programador[90%]
37	Evaluación F.G.	5 días	mié 02/06/10	mar 08/06/10	Probador
38	Reunión de seguimiento	2 horas	mar 08/06/10	mar 08/06/10	Jefe Proyecto;Analista
39	**Fase de elementos extras**	**27 días**	**jue 09/09/10**	**vie 15/10/10**	
40	**Análisis de requisitos F.E.E.**	**10 días**	**jue 09/09/10**	**mié 22/09/10**	
41	Efectos de sonido	2 días	jue 16/09/10	vie 17/09/10	Artista sonido
42	Gestión de almacenamiento de puntuacione	3 días	lun 20/09/10	mié 22/09/10	Analista
43	Diseño F.E.E.	5 días	jue 23/09/10	mié 29/09/10	Analista
44	Implementación F.E.E.	10 días	jue 30/09/10	mié 13/10/10	Programador[90%]
45	Evaluación F.E.E.	2 días	jue 14/10/10	vie 15/10/10	Probador
46	Reunión de seguimiento	2 horas	vie 15/10/10	vie 15/10/10	Jefe Proyecto;Analista
47	Evaluación y pruebas finales	10 días	lun 18/10/10	vie 29/10/10	Probador
48	Reunión final del proyecto	2 horas	vie 29/10/10	vie 29/10/10	Jefe Proyecto;Analista

Figura 87. Tareas del diagrama de GANTT inicial

El siguiente diagrama muestra las tareas de comienzo y fin correspondientes al diagrama de GANTT real; además de los recursos asignados a cada una de ellas antes del comienzo del proyecto.

Diagrama de Gantt

	Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
1	**Fase de formación previa**	**30 días**	**lun 21/09/09**	**vie 30/10/09**	
2	Estudio funcional sobre la creación de videojuegos	10 días	lun 21/09/09	vie 02/10/09	Programador[90%]
3	Formación en XNA	20 días	lun 05/10/09	vie 30/10/09	Programador[90%]
4	Reunión de seguimiento	2 horas	vie 30/10/09	vie 30/10/09	Analista;Jefe Proyecto;Programador
5	Desarrollo de la memoria del proyecto	275 días	lun 02/11/09	vie 19/11/10	Programador[10%]
6	**Fase estructural del proyecto**	**40 días**	**lun 02/11/09**	**vie 25/12/09**	
7	**Análisis de requisitos F.E.**	**20 días**	**lun 02/11/09**	**vie 27/11/09**	
8	Establecer los elementos principales	6 días	lun 02/11/09	lun 09/11/09	Analista
9	Establecer los movimientos principales	5 días	mar 10/11/09	lun 16/11/09	Analista
10	Eventos de las pantallas	4 días	mar 17/11/09	vie 20/11/09	Analista
11	Interrupciones del juego	5 días	lun 23/11/09	vie 27/11/09	Analista
12	Diseño F.E.	7 días	lun 30/11/09	mar 08/12/09	Analista
13	Implementación F.E.	10 días	mié 09/12/09	mar 22/12/09	Programador[90%]
14	Evaluación F.E.	1 día	mié 23/12/09	mié 23/12/09	Probador
15	Reunión de seguimiento	2 horas	mié 23/12/09	mié 23/12/09	Jefe Proyecto;Analista
16	**Fase de creación del juego**	**105 días**	**lun 11/01/10**	**vie 04/06/10**	
17	**Análisis de requisitos F.C.**	**40 días**	**lun 11/01/10**	**vie 05/03/10**	
18	Diseño gráfico y creación de los elementos	5 días	lun 11/01/10	vie 15/01/10	Artista gráfico
19	Aplicar movimientos a los elementos	10 días	lun 18/01/10	vie 29/01/10	Analista
20	Detección de colisiones	10 días	lun 01/02/10	vie 12/02/10	Analista
21	Algoritmos	10 días	lun 15/02/10	vie 26/02/10	Analista
22	Niveles y puntuaciones	5 días	lun 01/03/10	vie 05/03/10	Analista
23	Reunión de seguimiento	2 horas	vie 05/03/10	vie 05/03/10	Jefe Proyecto;Analista
24	Diseño F.C.	14 días	lun 08/03/10	jue 25/03/10	Analista
25	**Implementación F.C.**	**40 días**	**lun 29/03/10**	**vie 21/05/10**	**Programador[90%]**
26	Aplicar movimientos a los elementos	10 días	lun 29/03/10	vie 09/04/10	
27	Algoritmos	20 días	lun 12/04/10	vie 07/05/10	
28	Niveles y puntuaciones	10 días	lun 10/05/10	vie 21/05/10	
29	Evaluación F.C.	10 días	lun 24/05/10	vie 04/06/10	Probador
30	Reunión de seguimiento	2 horas	vie 04/06/10	vie 04/06/10	Jefe Proyecto;Analista
31	**Fase de gestión de pantallas**	**30 días**	**lun 07/06/10**	**vie 16/07/10**	
32	**Análisis de requisitos F.G.**	**10 días**	**lun 07/06/10**	**vie 18/06/10**	
33	Diseño gráfico	5 días	lun 07/06/10	vie 11/06/10	Artista gráfico
34	Navegación por pantallas	5 días	lun 14/06/10	vie 18/06/10	Analista;Artista gráfico
35	Diseño F.G.	5 días	lun 21/06/10	vie 25/06/10	Analista
36	Implementación F.G.	10 días	lun 28/06/10	vie 09/07/10	Programador[90%]
37	Evaluación F.G.	5 días	lun 12/07/10	vie 16/07/10	Probador
38	Reunión de seguimiento	2 horas	vie 16/07/10	vie 16/07/10	Jefe Proyecto;Analista
39	**Fase de elementos extras**	**22 días**	**jue 09/09/10**	**vie 08/10/10**	
40	**Análisis de requisitos F.E.E.**	**5 días**	**jue 09/09/10**	**mié 15/09/10**	
41	Efectos de sonido	2 días	jue 09/09/10	vie 10/09/10	Artista sonido
42	Gestión de almacenamiento de puntuacione	3 días	lun 13/09/10	mié 15/09/10	Analista
43	Diseño F.E.E.	5 días	jue 16/09/10	mié 22/09/10	Analista
44	Implementación F.E.E.	10 días	jue 23/09/10	mié 06/10/10	Programador[90%]
45	Evaluación F.E.E.	2 días	jue 07/10/10	vie 08/10/10	Probador
46	Reunión de seguimiento	2 horas	vie 08/10/10	vie 08/10/10	Jefe Proyecto;Analista
47	Evaluación y pruebas finales	10 días	lun 11/10/10	vie 22/10/10	Probador
48	Reunión final del proyecto	2 horas	vie 22/10/10	vie 22/10/10	Jefe Proyecto;Analista

Figura 88. Tareas del diagrama de GANTT real

Como se puede observar en las dos figuras anteriores, hay una pequeña diferencia en algunas de las fases en cuanto a estimación. Algunas tareas han tenido que ser replanificadas y a continuación se explica la razón de los desfases para cada una de ellas.

1. Desarrollo de la memoria del proyecto.

   Al comienzo del proyecto se estimaron 245 días al 10% de programador para el desarrollo de la memoria. Como la memoria se ha ido escribiendo a medida que se iba desarrollando el juego (ha estado viva durante todo el ciclo de vida del PFC), y el juego ha sufrido replanificaciones, la memoria se ha visto involucrada en el sentido de que se han ido añadiendo y modificando cosas hasta el punto de haberse alargado 30 días más, hasta los 275 días.

2. Implementación de la Fase Estructural del juego.

   La fase de implementación de la fase estructural ha sufrido un desvío de 2 días respecto a la planificación inicial. La causa principal fue que al finalizar la implementación de la estructura base del juego, el programador investigó la manera de optimizar el código, ya que al formar parte de la base del juego esta optimización provocaría una mejoría en el rendimiento del juego.

3. Análisis de Detección de colisiones.

   La detección de colisiones es una de las tareas más importantes del juego, por lo que la tarea de análisis se alargó 4 días más debido a la importancia de analizar, valorar y elegir el mejor método de detección de colisiones para este juego.

4. Análisis de Algoritmos.

   El análisis de los algoritmos empleados es otras de las tareas críticas del juego. Durante esta fase se analizaron todas las maneras posibles para realizar los algoritmos. Al tratarse de crear algoritmos propios, el análisis se vio afectado en 4 días más para lograr obtener los mejores algoritmos posibles.

5. Implementación de Algoritmos.

   Al igual que el análisis de los algoritmos fue una tarea ardua, la implementación no lo fue menos. El programador necesitó alargar 5 días más la implementación

de los algoritmos hasta conseguir las funciones óptimas de los algoritmos empleados.

6. Fase de Elementos Extras.

   Realmente durante esta fase las tareas se realizaron en el tiempo estimado, sin embargo, en la planificación inicial entre el diseño y la planificación de esta fase estaba la temporada estival donde el proyecto paró por vacaciones.

En la planificación real, las vacaciones empezaron al acabar la fase de generación de pantallas, y por tanto la fase de elementos extra se pudo realizar en 22 días, respecto a los 69 días reflejados en la planificación inicial.

En este proyecto, la diferencia en días respecto a la planificación inicial respecto a la final se ha visto alargada principalmente en la fase de creación de la memoria, ya que la decisión de hacer una memoria de calidad ha conllevado mayor tiempo de investigación y de elaboración.

El desfase en las tareas es un desfase mínimo teniendo en cuenta la dificultad del proyecto y los problemas encontrados durante su elaboración, comentados en apartados anteriores, por lo que puede concluirse que la planificación inicial desarrollada por el jefe de proyecto se adaptado casi totalmente a la realidad del proyecto.

En las próximas figuras, se mostrará la duración de cada tarea en el diagrama de GANTT de las planificaciones real e inicial indicando su evolución en el tiempo y las dependencias para ver las más claramente las diferencias entre lo planificado inicialmente y lo que se ha conseguido.

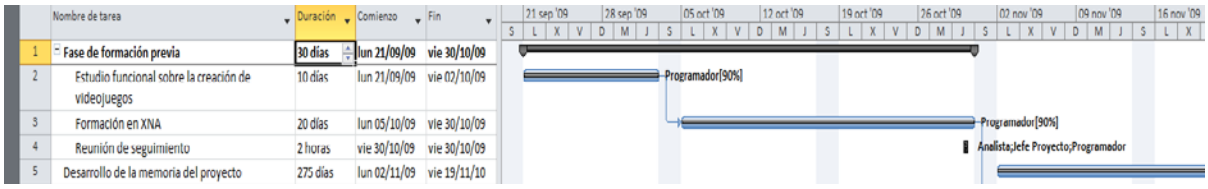

Figura 89. Diagrama de GANTT real de fase formación previa

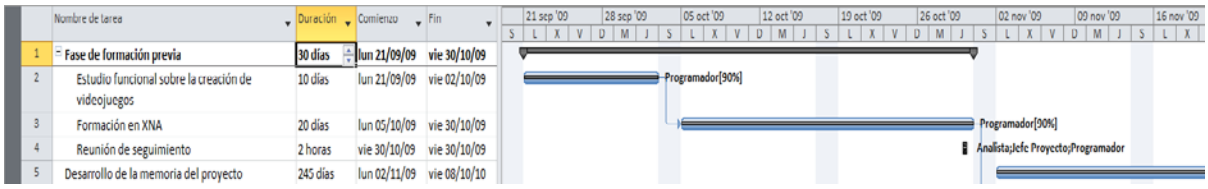

Figura 90. Diagrama de GANTT inicial de fase formación previa

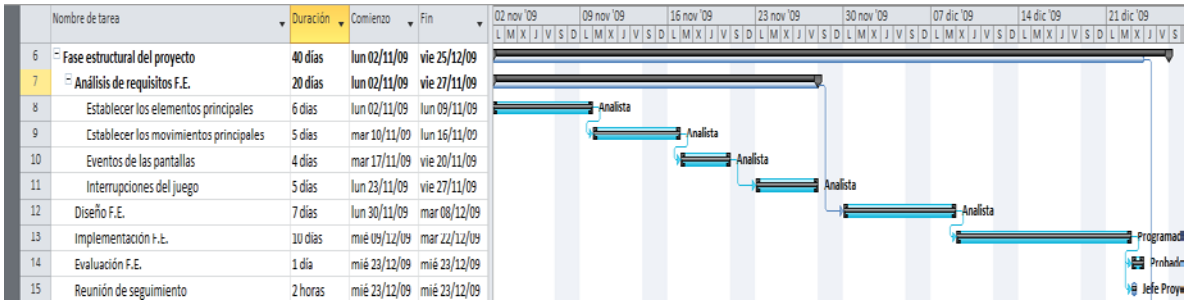

Figura 91. Diagrama de GANTT real de fase estructural del proyecto

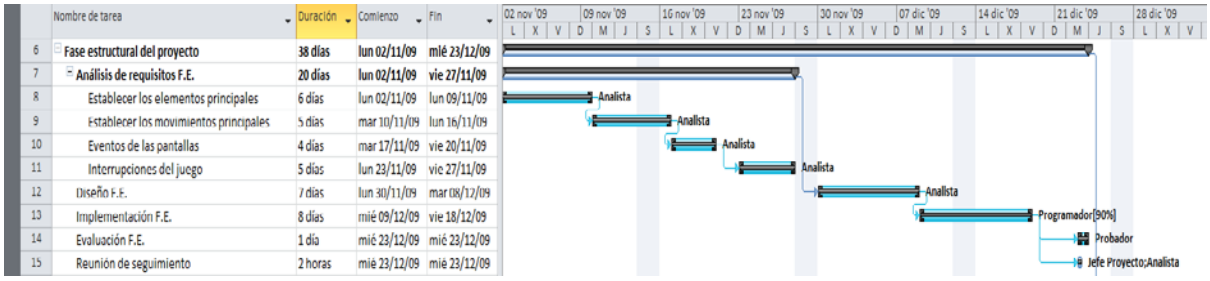

Figura 92. Diagrama de GANTT inicial de fase estructural del proyecto

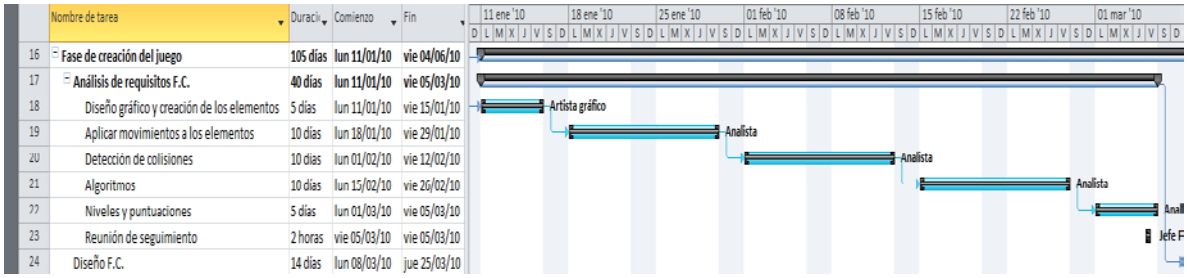

Figura 93. Diagrama de GANTT real de fase de creación del juego I

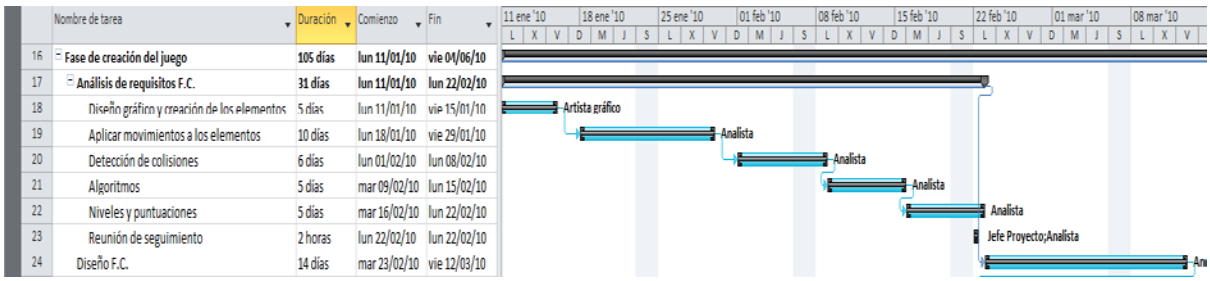

Figura 94. Diagrama de GANTT inicial de fase de creación del juego I

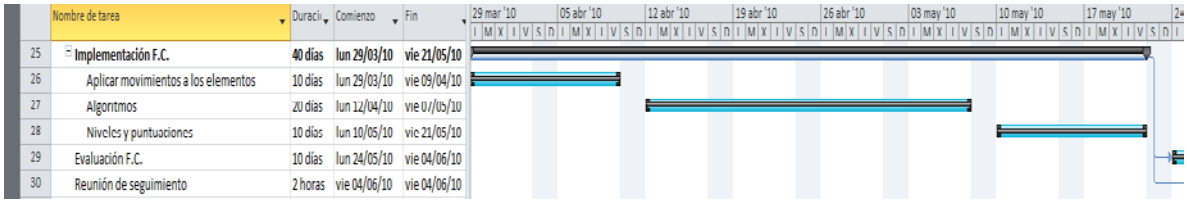

Figura 95. Diagrama de GANTT real de fase de creación del juego II

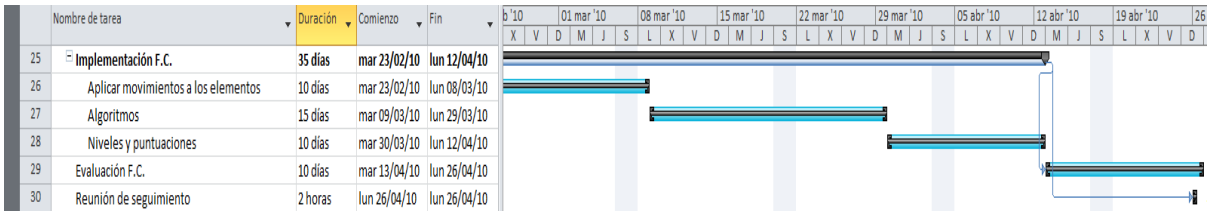

Figura 96. Diagrama de GANTT inicial de fase de creación del juego II

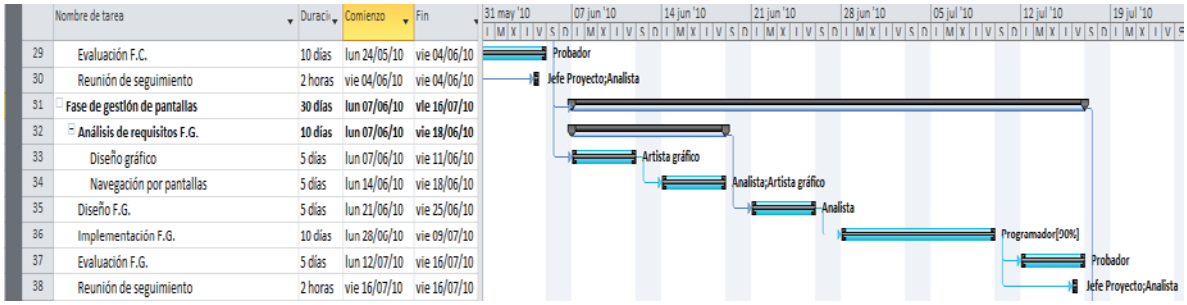

Figura 97. Diagrama de GANTT real de fase de gestión de pantallas

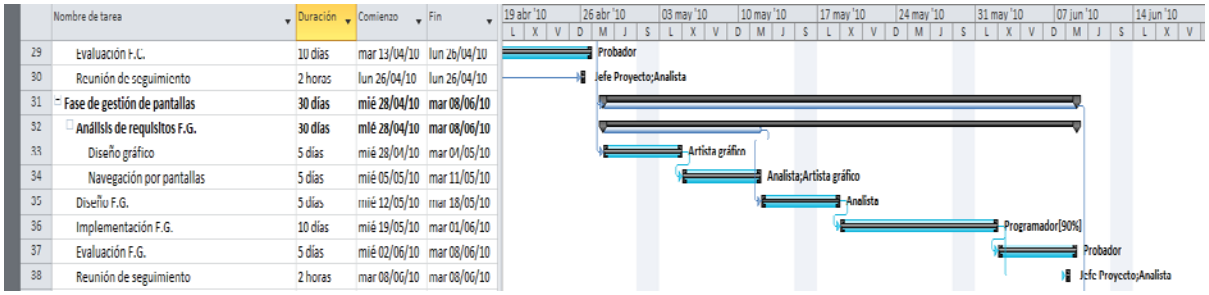

Figura 98. Diagrama de GANTT inicial de fase de gestión de pantallas

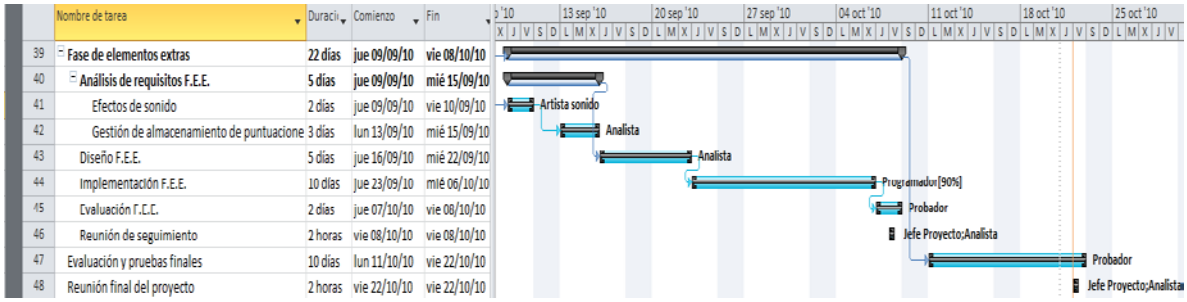

Figura 99. Diagrama de GANTT real de fase de elementos extra

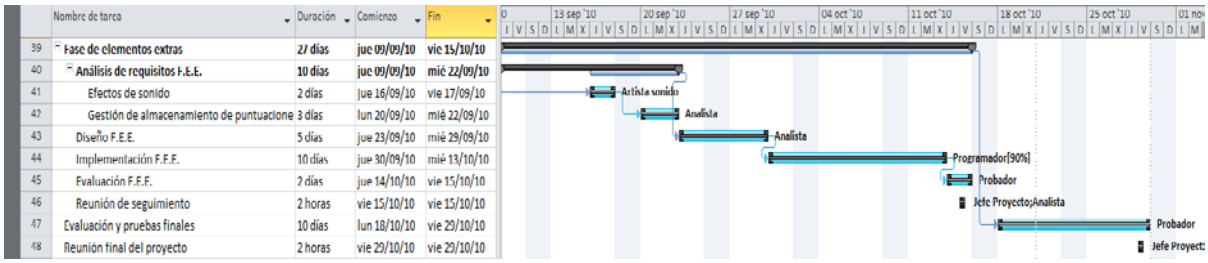

Figura 100. Diagrama de GANTT inicial de fase de elementos extra

Una vez estudiados los diagramas de GANTT inicial y real, se puede determinar que la planificación inicial, a pesar de haber sufrido alguna desviación en el tiempo respecto a la planificación real se adapta bastante bien al trabajo realizado, por lo que los plazos de entrega de un proyecto de casi un año de duración son mínimos en cuanto a coste se refiere, ya que las tareas se han podido realizar prácticamente en los plazos determinados al comienzo del proyecto.

A4. Presupuesto del proyecto

A continuación se hará una estimación del proyecto en función de los recursos humanos necesarios para llevarlo a cabo. El presupuesto es un factor clave para el estudio de la viabilidad. El presupuesto se realiza asumiendo unas fechas y condiciones determinadas las cuales son similares a las que realmente se va a encontrar quien realiza el proyecto durante el desarrollo del mismo. No se debe olvidar que cualquier variación de estas condiciones reales respecto de lo planeado puede suponer una modificación importante del coste inicialmente previsto.

A.4.1. Estimación

La estimación recoge de forma ordenada los recursos necesarios para la ejecución del proyecto, ya sean recursos materiales o recursos humanos. En el caso de los recursos humanos, se proporciona el número de horas de trabajo de cada persona necesarias para llevar a cabo la ejecución del proyecto, y en los materiales las unidades. La siguiente tabla muestra los datos correspondientes.

RECURSO HUMANO	TRABAJO
Jefe de Proyecto	14 horas
Analista	370 horas
Artista gráfico	60 horas
Artista sonido	8 horas
Programador	486 horas
Probador	112 horas
RECURSO MATERIAL	UNIDADES
PC con Windows 7	1
MS Visual Studio 2008	1
MS Office 2010	1

Tabla 5. Recursos y horas asociados al proyecto

Además se debe tener en cuenta el precio de cada recurso, reflejando en la estimación el coste de la hora de trabajo del empleado. La siguiente tabla hace referencia a estos datos.

RECURSO HUMANO	€/Hora	€ Total
Jefe de Proyecto	23,08	323,12
Analista	16,71	6516,90
Artista gráfico	9,31	558,60
Artista sonido	9,31	74,48
Programador	10,05	4844,10
Probador	10,02	1122,24
RECURSO MATERIAL		€/ Unidad
PC con Windows 7		700
MS Visual Studio 2008		550
MS Office 2010		140

Tabla 6. Costes de los recursos asociados al proyecto

A.4.2. Presupuesto final

Supone la valoración definitiva del coste del proyecto. A continuación se muestra una tabla en la que se recoge el coste asociado a cada fase del proyecto y el coste total.

Fases del proyecto	Coste €
Fase de formación previa	1185,08
Desarrollo de la memoria	1206,00
Fase estructural	2286,14
Fase de creación del juego	5468,52
Fase de gestión de pantallas	1682,58
Fase de elementos extras	1130,74
Evaluación y pruebas finales	400,80
Reunión final del proyecto	79,58
Total Fases	**13439,44**
Total Recursos Materiales	**1390,00**
Total del Proyecto	**14829,44**

Tabla 7. Presupuesto final del proyecto

El coste total del proyecto, incluidos los Recursos Humanos y Recursos Materiales ascienden a un total de 14829,44€ en un total de 300 días.

A5. Publicación del juego

En este punto se analizará la opción de publicar el juego y la manera de rentabilizarlo. Para ello, lo primero que hay que hacer es decidir dónde publicarlo. Como se trata de un juego desarrollado en XNA, la mejor opción es publicarlo en el bazar de XBOX 360 puesto que con el pago de la suscripción de *XNA Creators Club*[38] es posible colgar todos los juegos que se desee en el bazar durante un año, proporcionando una forma relativamente rápida de distribución.

Una vez que el juego XNA ha sido desarrollado, hay que enviarlo a los comprobadores o *testers* de Microsoft para su revisión. La revisión suele consistir en comprobar que el juego esté correctamente etiquetado, y los datos que hay que enviar para la revisión son los siguientes:

- Información del juego.
- Un binario del juego.
- Países en los que se publicará.
- Posibles comentarios.
- Indicación de si el juego es una versión total o una demo.

Los juegos con archivos binarios inferiores a 50 MB de tamaño se pueden adquirir con 80, 240, 400 u 800 puntos. Los juegos con archivos binarios iguales o mayores a 50 MB de tamaño se pueden adquirir con 240, 400 u 800 puntos.

Una vez que el juego ha sido revisado y validado pasa a formar parte del bazar y puede ser descargado, momento en el cual el juego puede comenzar a generar beneficios. El sistema de beneficios económicos sigue una normativa basada en puntos y unidades vendidas. El creador empezará a ganar dinero una vez haya vendido un número de copias establecidas por el sistema de puntos de Microsoft, y todo lo ganado durante estas primeras copias va a las arcas de Microsoft, además de un porcentaje por cada copia vendida a partir del número de copias mínimas vendidas en concepto de derechos de promoción.

El impuesto promocional exacto será de entre un 10 y un 30 por ciento para todos los juegos, aunque no hay establecido un porcentaje exacto, ya que se fija el primer trimestre de cada año. Por otro lado, el límite de pago mínimo que establece Microsoft es de 150 dólares estadounidenses. Si no se alcanza este mínimo, no se ganará nada. Según el precio que se elija para el juego, el mínimo se establecerá por cuatrimestres, semestres o por años.

Teniendo en cuenta estos factores, lo recomendable es fijar un valor inicial de 240 *Microsoft Points* (equivalentes a 2,40 dólares) para el primer semestre y posteriormente reducirlo a 80 Microsoft Points (equivalentes a 1 dólar) una vez que el juego haya alcanzado algo de fama en la comunidad.

Por otro lado se puede utilizar el sistema de ***Tokens***[39] (códigos para adquirir el juego de manera gratuita) para hacer publicidad en revistas y blogs gastando lo mínimo posible.

Según los datos de *Microsoft*, cerca del 70% de los jugadores de *XBOX 360* descargan juegos arcade y la media de descargas por usuario es de 7. En cuanto a los datos de descarga por juego, la

media de un juego de éxito es de 350.000 en los dos primeros meses, recuperando un 156% de la inversión en el primer año.

Según estos datos, el cálculo mínimo de descargas necesarias para rentabilizar el juego en un periodo de dos años (cuatro semestres), teniendo en cuenta que el coste total del juego ha sido de **14829,44€** equivalente a **20761,21$**. Para realizar unos cálculos aproximados y pretendiendo recuperar un cuarto de la inversión en cada semestre, es decir, **5190,3 $**, los cálculos serían los siguientes:

- Durante el primer semestre el juego tiene un valor de 240 *Microsoft Points* (equivalentes a **2,4$**)
    - Para superar el límite de pago mínimo de 150$ se deben realizar **63** descargas, de las que no se obtiene nada.
    - Una vez superado el límite de pago mínimo, del resto de copias se debe restar aproximadamente un 15% del valor que se queda Microsoft en derechos de promoción. De cada copia, restando este porcentaje, se obtendrán por tanto **2,04$**.
    - Para alcanzar los 5190,3$ del semestre deben producirse por tanto **2545** descargas.

- A partir del segundo semestre, el juego costará 80 *Microsoft Points* (equivalentes a **1$**)
    - Para superar el límite de pago mínimo de 150$ y restando el 15% de derechos de cada copia, deben realizarse un total de **177** descargas.
    - Para alcanzar los 5190,3$ deben producirse **6107** descargas.

Ésto es sólo una estimación de cómo se rentabilizaría en juego en dos años. Las cifras varían bastante si el número de copias aumenta el primer semestre, puesto que el precio del juego es casi el triple en comparación con los semestres posteriores. De hecho según Microsoft, los primeros dos meses se suele recuperar el 35% de la inversión total en un juego arcade. De modo que, en el primer semestre, se podría recuperar gran parte de la inversión realizada obteniendo beneficios mucho antes.

Diccionario de términos y acrónimos

[1] **Videojuego**. Dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor o de un ordenador. [↑]

[2] **API**. Un interfaz de programación de aplicaciones o API (del inglés *application programming interface*) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. [↑]

[3] **Microsoft XNA**. *Microsoft XNA* es un conjunto de herramientas con un entorno de ejecución administrado proporcionado por *Microsoft* que facilita el desarrollo de videojuegos compatibles con las plataformas *XBOX 360,* PC y *Zune*. [↑]

[4] **Xbox 360**. Xbox 360 es la segunda videoconsola de sobremesa producida por Microsoft desarrollada en colaboración con IBM y ATI. [↑]

[5] **Bubble shooter**. Adictivo y popular juego de puzzle que es excelente para niños y adultos. La premisa es lanzar bolas del mismo color para formar grupos de 3 o más para salvarlos. El juego termina cuando las bolas llenan la pantalla y una toca el fondo. Además, este juego permite ajustar el nivel para encajar con la habilidad del jugador. [↑]

[6] **Consola**. Dispositivo que, integrado o no en una máquina, contiene los instrumentos para su control y operación. [↑]

[7] **Sprite**. Se trata de un tipo de mapa de bits dibujados en la pantalla de un ordenador por hardware gráfico especializado sin cálculos adicionales de la CPU. A menudo son pequeños y parcialmente transparentes. [↑]

[8] **EDSAC**. Acrónimo de ***E**lectronic **D**elay **S**torage **A**utomatic **C**alculator*, fue una antigua computadora británica. La EDSAC fue el primer calculador electrónico en el mundo en contar con órdenes internas, aunque no la primera computadora con programas internos. [↑]

[9] **Aventuras gráficas**. Subgénero de los videojuegos de aventuras. La dinámica de este tipo de juego consiste en ir avanzando por el mismo a través de la resolución de diversos rompecabezas, planteados como situaciones que se suceden en la historia, interactuando con personajes y objetos a través de un menú de acciones o

interfaz similar, utilizando un cursor para mover al personaje y realizar las distintas acciones. El concepto clásico de aventura gráfica incluía la visión de los personajes en tercera persona la mayor parte del juego, aunque algunas de las aventuras gráficas más importantes se planteasen en primera persona. [↑]

[10] **Arcade**. Arcade es el término genérico de las máquinas recreativas de videojuegos o también llamadas "*maquinitas*" disponibles en lugares públicos de diversión, centros comerciales, restaurantes, bares, o salones recreativos especializados. [↑]

[11] **PCM**. La modulación por impulsos codificados **(MIC** o ***PCM*** por sus siglas inglesas de *Pulse Code Modulation*) es un procedimiento de modulación utilizado para transformar una señal analógica en una secuencia de bits (señal digital). Una trama PCM es por tanto una representación digital de una señal analógica en donde la magnitud de la onda análogica es tomada en intervalos uniformes (muestras), donde cada muestra puede tomar un conjunto finito de valores, los cuales se encuentra codificados. [↑]

[12] **Multijugador**. Se trata de un modo de videojuego en el que participan diversos jugadores. Más concretamente se suele utilizar para definir videojuegos que se juegan a través de Internet u otro tipo red, con otras personas conectadas a la misma. [↑]

[13] **DirectX**. DirectX es una colección de API creadas y recreadas para facilitar las complejas tareas relacionadas con multimedia, especialmente programación de juegos y vídeo en la plataforma Microsoft Windows. [↑]

[14] **MMOG**. MMOG viene de las siglas en inglés *massively multiplayer online game*. Se trata de un videojuego en donde pueden participar e interactuar en un mundo virtual, un gran número de jugadores simultáneamente (multijugador) conectados a través de la red (en línea), normalmente Internet dado el grado de concurrencia que pueden llegar a alcanzar y las características técnicas de los servidores que han de gestionar ese volumen de conexiones. [↑]

[15] **GANTT**. El diagrama de Gantt, gráfica de Gantt o carta Gantt es una popular herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. A pesar de que, en principio, el diagrama de Gantt no indica las relaciones existentes entre actividades, la posición de cada tarea a lo largo del tiempo hace que se puedan identificar dichas relaciones e interdependencias. [↑]

[16] **PERT**. La Técnica de Revisión y Evaluación de Programas (en inglés *Program Evaluation and Review Technique*), comúnmente abreviada como **PERT**, es un modelo para la administración y gestión de proyectos inventado en 1958 por la Oficina de Proyectos Especiales de la Marina de Guerra del Departamento de Defensa de los EE. UU. PERT es básicamente un método para analizar las tareas involucradas en completar un proyecto dado, especialmente el tiempo para completar cada tarea, e identificar el tiempo mínimo necesario para completar el proyecto total. [↑]

[17] **.NET**. Se trata de un framework de Microsoft que hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones. Basado en ella, la empresa intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el sistema operativo hasta las herramientas de mercado. [↑]

[18] **DLL**. Nombre común que reciben las bibliotecas de enlace dinámico. Es el término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. Esta denominación es exclusiva a los sistemas operativos Windows siendo "*.dll*" la extensión con la que se identifican estos ficheros, aunque el concepto existe en prácticamente todos los sistemas operativos modernos. [↑]

[19] **Multiplataforma**. Término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en uno x86 (solo para equipos Apple). [↑]

[20] **Renderización**. Es un término usado en informática para referirse al proceso de generar una imagen desde un modelo. Este término técnico es utilizado por los animadores o productores audiovisuales y en programas de diseño en 3D. En términos de visualizaciones en un ordenador, más específicamente en 3D, la renderización es un proceso de cálculo complejo desarrollado por un ordenador destinado a generar una imagen 2D a partir de una escena 3D. La traducción más fidedigna es *interpretación*, aunque se suele usar el término inglés. Así podría decirse que en el proceso de renderización, el ordenador *interpreta* la escena en tres dimensiones y la plasma en una imagen bidimensional. [↑]

[21] **IDE**. Un IDE (en inglés *integrated development environment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir,

consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDE's pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. [↑]

[22] **Tao Framework**. Librería que permite utilizar las funciones de diversas librerías libres, enfocadas principalmente a la programación de gráficos y de videojuegos, con la gran ventaja de que es realmente multiplataforma, permitiendo escribir programas que funcionarán tanto en Windows, como en Linux o en MacOS. [↑]

[23] **.NET Compact Framework**. Microsoft .NET Compact Framework es un componente integral de los dispositivos Windows Mobile y Windows Embedded CE que permite generar y ejecutar aplicaciones administradas y utilizar servicios web. .NET Compact Framework incluye un Common Language Runtime (CLR) optimizado y un subconjunto de la biblioteca de clases de .NET Framework, que admite características como Windows Communication Foundation (WCF) y formularios Windows Forms. También contiene clases que están diseñadas exclusivamente para .NET Compact Framework. [↑]

[24] **XACT**. Es un programa realizado por Apple que básicamente contiene múltiples utilidades de audio del mundo UNIX con una GUI bastante sencilla con la que es posible acceder a múltiples utilidades rápidamente, suponiendo un ahorro de tiempo importante. [↑]

[25] **Gamepad**. Dispositivo de entrada usado para interactuar con un videojuego ya sea para consola o PC. El gamepad o control de mando permite moverse e interactuar con los elementos del juego para realizar las diversas acciones necesarias para cumplir los objetivos. [↑]

[26] **LGPL**. La Licencia Pública General Reducida de GNU, o más conocida por su nombre en inglés *GNU Lesser General Public License* (antes *GNU Library General Public License* o Licencia Pública General para Bibliotecas de GNU), o simplemente por su acrónimo del inglés GNU LGPL es una licencia de software creada por la *Free Software Foundation*. Esta licencia pública general se aplica a la mayoría del software de la Fundación para el software libre y a cualquier otro programa de software cuyos autores así lo establecen. [↑]

[27] **Skeletal**. También conocido como "*Skeletal Mesh*", es un modelo dinámico de animación basado en los huesos, es decir, todos estos modelos representados tienen una base o esqueleto que determina sus movimientos. [↑]

[28] **Morph**. Es un modelo dinámico de animación basado en puntos de inflexión, utilizado principalmente para animaciones faciales y de expresión. [↑]

[29] **Environment mapping**. Es una forma de mapeado de texturas en la cual las coordenadas de la textura son dependientes de la vista. Por ejemplo, simular reflejo en un objeto con brillo. [↑]

[30] **Stencil buffer**. Técnica aplicada para generar sombras y brillos en aplicaciones en 3D basada en el uso de buffers de memoria para procesar la información de los píxeles en tiempo real. [↑]

[31] **Modder**. Un mod en informática es cualquier tipo de cambio a algún programa, mejorándolo o modificándolo respecto a la forma original del mismo. Un modder es aquel que realiza dicho cambio. En los videojuegos, sobre todo en los de PC, son muy típicos los mods que modifican la funcionalidad del juego, por ejemplo añadiendo nuevas capacidades a los personajes. [↑]

[32] **UML**. Lenguaje Unificado de Modelado (**UML**, del inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "*plano*" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. [↑]

[33] **Diagramas de casos de uso**. En UML, un diagrama de casos de uso es un diagrama de comportamiento. Los diagramas de casos de uso son a menudo confundidos con los casos de uso. Mientras los dos conceptos están relacionados, los casos de uso son mucho más detallados que los diagramas de casos de uso. [↑]

[34] **Shader**. La tecnología *shaders* es cualquier unidad escrita en un lenguaje de sombreado que se puede compilar independientemente. Es una tecnología reciente y que ha experimentado una gran evolución destinada a proporcionar al programador una interacción con la Unidad de Procesamiento Gráfico o GPU (procesador dedicado al procesamiento de gráficos) hasta ahora imposible. Los shaders son utilizados para realizar transformaciones y crear efectos especiales, como por ejemplo iluminación, fuego o niebla. Para su programación los shaders utilizan lenguajes específicos de alto nivel que permitan la independencia del hardware. [↑]

[35] **Frame**. Se denomina frame en inglés, a un fotograma o cuadro, una imagen particular dentro de una sucesión de imágenes que componen una animación. La continua sucesión de estos fotogramas producen a la vista la sensación de movimiento, fenómeno dado por las pequeñas diferencias que hay entre cada uno de ellos. [↑]

[36] **Pixel**. Un píxel (acrónimo del inglés *picture element*, "*elemento de imagen*") es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de vídeo o un gráfico. [↑]

[37] **Algoritmo**. un algoritmo es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución. [↑]

[38] **XNA Creators Club.** *XNA Creators Club* es una plataforma online creada por *Microsoft* para dar apoyo a los programadores de juegos en *XNA*. A través de la plataforma se proporcionan tutoriales, ejemplos de buenas prácticas, foros de discusión etc. Para poder publicar un juego en el bazar online de *XBOX 360* es necesario ser miembro *Premium* de *XNA Creators Club*, lo que supone una cuota de 49$ por un cuatrimestre o 99$ por un año. [↑]

[39] **Token**. En el ámbito de *XBOX 360*, un token es un código único que permite a su poseedor descargar un determinado contenido son gastar *Microsoft Points*. El contenido descargado suele asociarse con el número de serie de la consola por lo que no podrá ser distribuido evitando la copia. Los token suelen utilizarse para distribuir betas de juegos a revistas o blogs especializados o para proporcionar contenidos exclusivos a los usuarios. [↑]

Referencias

[1] Bubble Shooter – Bubble Shooter. El juego favorito del mundo [en línea]. [ref. de 2009]. Disponible en World Wide Web: < http://bubbleshooter.es/ > [↑]

[2] Campus usual – Videojuegos y educación. Psicología del aprendizaje [en línea]. [ref de 1995]. Disponible en World Wide Web: <http://campus.usal.es/~teoriaeducacion/rev_numero_02/n2_art_etxeberria.htm > [↑]

[3] Diccionario de la lengua española – Real Academia Española [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://www.rae.es/rae.html > [↑]

[4] El otro lado – Historia de los videojuegos: Prehistoria [en línea]. [ref. de 2008]. Disponible en World Wide Web: <http://www.elotrolado.net/wiki/Prehistoria_de_los_Videojuegos > [↑]

[5] Índice latino – Historia de los videojuegos: Orígenes [en línea]. [ref. de 2008]. Disponible en World Wide Web: <http://indicelatino.com/juegos/historia/origenes/> [↑]

[6] Full blog – Historia de los videojuegos: Historia de los videojuegos. [en línea]. [ref. de 2008]. Disponible en World Wide Web: <http://thenewgamer10.fullblog.com.ar/post/la-historia-de-los-videojuegos-decadas-de-los-70-111211662230/ > [↑]

[7] Gazcue es arte – El Atari, primera consola de videojuegos. [en línea]. [ref. de 2008]. Disponible en World Wide Web: <http://gazcueesarte.blogspot.com/2008/06/el-atariprimera-consola-de-videos.html > [↑]

[8] Wikipedia, la enciclopedia libre – Atari [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Atari > [↑]

[9] Wikipedia, la enciclopedia libre – Nintendo [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Nintendo > [↑]

[10] Wikipedia, la enciclopedia libre – Sega [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Sega > [↑]

[11] Funversion – Videojuegos, la revolución del entretenimiento. [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://funversion.universia.es/videojuegos/reportaje/30anyosconsolas.jsp > [↑]

[12] Wikipedia, la enciclopedia libre – ZX Spectrum [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/ZX_Spectrum > [↑]

[13] Wikipedia, la enciclopedia libre – Amstrad CPC [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/CPC_464 > [↑]

[14] Wikipedia, la enciclopedia libre – Donkey Kong [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Donkey_Kong > [↑]

[15] Likantropia – Haunted House, un juego de rol [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://www.likantropia.com/haunted/ > [↑]

[16] Lo que el tiempo se llevó – Mario Bros [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://eltiemposellevo.blogspot.com/ > [↑]

[17] EA – EA Videojuegos [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://www.ea.com/es > [↑]

[18] Geekets – La edad de oro del software español [en línea]. [ref. de 2008]. Disponible en World Wide Web: <http://www.geekets.com/2008/08/la-edad-de-oro-del-software-espanol-i-dinamic/ > [↑]

[19] Mural UV – King's Quest [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://mural.uv.es/abelgar/inventario/kquest.html > [↑]

[20] MicroCaos – Historia del Tetris [en línea]. [ref. de 2007]. Disponible en World Wide Web: < http://www.microcaos.net/juegos/videojuegos/tetris-historia-del-tetris/ > [↑]

[21] El Otro Lado – Historia de los Videojuegos [en línea]. [ref. de 2008]. Disponible en World Wide Web: < http://www.elotrolado.net/wiki/Historia_de_los_videojuegos > [↑]

[22] Abadia del crimen – Página oficial de Abadía del crimen [en línea]. [ref. de 2008]. Disponible en World Wide Web: < http:// www.abadiadelcrimen.com/ > [↑]

[23] Telepolis – Distrito Sims [en línea]. [ref. de 2008]. Disponible en World Wide Web: <http://www.telepolis.com/cgi-bin/web/DISTRITODOCVIEW?url=/1582/doc/eljuego/creador.htm > [↑]

[24] Wikipedia, la enciclopedia libre – Neo Geo [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Neo-Geo >[↑]

[25] Wikipedia, la enciclopedia libre – Sega Saturn [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Sega_Saturn > [↑]

[26] Wikipedia, la enciclopedia libre – Dream Cast [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Dreamcast > [↑]

[27] El Otro Lado – Historia de los Videojuegos: Década de los 90 [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://www.elotrolado.net/wiki/Historia_de_los_videojuegos > [↑]

[28] Wikipedia, la enciclopedia libre – Mortal Kombat [en línea]. [ref. de 2010]. Disponible en World Wide Web:<http://es.wikipedia.org/wiki/Mortal_Kombat_%28serie%29 > [↑]

[29] Wikipedia, la enciclopedia libre – Alone in the Dark [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Alone_in_the_Dark > [↑]

[30] Wikipedia, la enciclopedia libre – Doom [en línea]. [ref. de 2010]. Disponibleen World Wide Web: < http://es.wikipedia.org/wiki/Doom > [↑]

[31] Mediotiempo – Museo FIFA [en línea]. [ref. de 209]. Disponible en World Wide Web: <http://www.mediotiempo.com/videojuegos/fifa09/noticias/2009/04/13/museo-fifa-asi-se-veia-el-fifa-94-el-primero-de-la-historia > [↑]

[32] Wikijuegos – Resident Evil [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://videojuego.wikia.com/wiki/Resident_Evil > [↑]

[33] Wikijuegos – Metal Gear Solid [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://videojuego.wikia.com/wiki/Metal_Gear_Solid > [↑]

[34] Wikijuegos – StarCraft [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://videojuego.wikia.com/wiki/StarCraft > [↑]

[35] Wikipedia, la enciclopedia libre – Commandos [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Commandos > [↑]

[36] Wikipedia, la enciclopedia libre – Nintendo DS [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Nintendo_DS > [↑]

[37] Website oficial de PlayStation – PlayStation 3 [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.playstation.com/ps3 > [↑]

[38] Wikipedia, la enciclopedia libre – Halo [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Universo_de_Halo > [↑]

[39] Crónicas de un verdugo – World of Warcraft [en línea]. [ref. de 2008]. Disponible en World Wide Web: < http://www.carlosverdugo.cl/content/view/121044/WORLD-OF-WARCRAFT-ALCANZA-UN-NUEVO-HITO-10-MILLONES-DE-SUSCRIPTORES.html > [↑]

[40] Wikipedia, la enciclopedia libre – God of War [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/God_of_War > [↑]

[41] Guitar Hero – Activision [en línea]. [ref. de 209]. Disponible en World Wide Web: <http://www.guitarhero.es/ > [↑]

[42] La flecha – Diario de ciencia y tecnología [en línea]. [ref. de 2006]. Disponible en World Wide Web: < http://www.laflecha.net/canales/videojuegos/noticias/200605111 > [↑]

[43] Xbox– Kinect [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://www.xbox.com/es-ES/kinect/ > [↑]

[44] Wikipedia, la enciclopedia libre – Modchip [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Modchip > [↑]

[45] Psjailbreak– Chipset [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://psjailbreak.com > [↑]

[46] On Live – Página oficial [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://www.onlive.com> [↑]

[47] GameOver – Realidad Virtual [en línea]. [ref. de 2007]. Disponible en World Wide Web: <http://www.gameover.es/gameover/el-futuro-de-los-videojuegos-la-realidad-virtual.html > [↑]

[48] Punto de partida – Vyirtual boy [en línea]. [ref. de 2010]. Disponible en World Wide Web:
<http://www.puntodepartida.com/retroinformatica/maquinadeltiempo/virtualboy.php>[↑]

[49] Activ@mente – Realidad Virtual [en línea]. [ref. de 2008]. Disponible en World Wide Web: < http://www.activamente.com.mx/vrml/ > [↑]

[50] Modulo Web – Second Life [en línea]. [ref. de 2008]. Disponible en World Wide Web:
<http://moduloweb.wikidot.com/comunidades-virtuales-y-metaversos > [↑]

[51] Wikipedia, la enciclopedia libre – Desarrollo de videojuegos [en línea]. [ref. de 2010]. Disponible en World Wide Web:
<http://es.wikipedia.org/wiki/Desarrollo_de_videojuegos > [↑]

[52] XNA Comunity – XNA [en línea]. [ref. de 2008]. Disponible en World Wide Web:
<http://xnacommunity.codeplex.com/wikipage?title=%u00bfQu%u00e9%20es%20XNA%3F > [↑]

[53] Winjanet – XNA Framework [en línea]. [ref. de 2008]. Disponible en World Wide Web:
<http://forum.winjanet.66ghz.com/index.php?topic=135.0 > [↑]

[54] Juank.black-byte – XNA Content Pipeline [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://juank.black-byte.com/xna-content-pipeline/ > [↑]

[55] Wikipedia, la enciclopedia libre – XNA Game Studio [en línea]. [ref. de 2010]. Disponible en World Wide Web:
<http://es.wikipedia.org/wiki/Microsoft_XNA#XNA_Game_Studio > [↑]

[56] Adventure Game Studio [en línea]. [ref. de 2010]. Disponible en World Wide Web:
<http://www.adventuregamestudio.co.uk/ > [↑]

[57] Build Engine [en línea]. [ref. de 2002]. Disponible en World Wide Web:
<http://advsys.net/ken/build.htm > [↑]

[58] Blender 3D [en línea]. [ref. de 2010]. Disponible en World Wide Web:
<http://www.blender.org/ > [↑]

[59] Crystal Space 3D [en línea]. [ref. de 2010]. Disponible en World Wide Web:
<http://www.crystalspace3d.org/ > [↑]

[60] Dim3 [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://dim3wiki.site88.net/ > [↑]

[61] Wikipedia, la enciclopedia libre – Dom Engine [en línea]. [ref. de 2010]. Disponible en World Wide Web: < http://es.wikipedia.org/wiki/Doom_Engine > [↑]

[62] Emergent – Gamebryo [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://www.emergent.net/ > [↑]

[63] 3D Game Studio [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://www.3dgamestudio.com/ > [↑]

[64] Irrlicht Engine [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://irrlicht.sourceforge.net/ > [↑]

[65] M.U.G.E.N. [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://mugenhispania.org/ > [↑]

[66] Quest3D [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://quest3d.com/> [↑]

[67] Valve Software – Source [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://source.valvesoftware.com/ > [↑]

[68] Valve Software – Source [en línea]. [ref. de 2010]. Disponible en World Wide Web: <http://source.valvesoftware.com/ > [↑]